รายงานวิจัยฉบับสมบูรณ์

โครงการสนับสนุนการออกแบบระบบปฏิบัติการที่สามารถขยายและยืดหยุ่นได้ด้วยโครงร่างเชิงลักษณะ

โดย ผู้ช่วยศาสตราจารย์ ดร.ปณิธิ เนตินันทน์

กรกฎาคม 2549

รายงานวิจัยฉบับสมบูรณ์

โครงการสนับสนุนการออกแบบระบบปฏิบัติการที่สามารถขยาย และยืดหยุ่นได้ด้วยโครงร่างเชิงลักษณะ

โดย ผู้ช่วยศาสตราจารย์ ดร.ปณิธิ เนตินันทน์

กรกฎาคม 2549

สัญญาเลขที่ MRG4780168

รายงานวิจัยฉบับสมบูรณ์

โครงการสนับสนุนการออกแบบระบบปฏิบัติการที่สามารถขยาย และยืดหยุ่นได้ด้วยโครงร่างเชิงลักษณะ

ผู้ช่วยศาสตราจารย์ ดร.ปณิธิ เนดินันทน์
มหาวิทยาลัยกรุงเทพ
คณะวิทยาศาสตร์และเทคโนโลยี
ภาควิชาวิทยาการคอมพิวเตอร์

สนับสนุนโดยสำนักงานคณะกรรมการการอุดมศึกษา
และสำนักงานกองทุนสนับสนุนการวิจัย

กิตติกรรมประกาศ

ขอกราบขอบพระคุณบิดาและมารดาของกระผม ที่ให้ความอนุเคราะห์และเป็นกำลังใจให้ดลอดเสมอมา ภรรยาผู้ร่วมสุขทุกข์เขียงข้างมา 4 ปีและบุตรที่น่ารัก อันเป็นแรงบันดาลใจให้วิริยะ อุตสาหะและมานะพากเพียรในการทำงานวิจัยชิ้นนี้ให้สำเร็จด้วยดีตลอด 2 ปีในการทำวิจัยนี้ ขอขอบพระคุณ โดยเฉพาะ ศาสตราจารย์การวิจัย (Research Professor) Tzilla Elrad แห่งสถาบัน Illinois Institute of Technology, Illinois, U.S.A. ที่คอยให้ความช่วยเหลือ ให้คำแนะนำการดำเนินการวิจัยและตรวจสอบบทความตีพิมพ์เผยแพร่ในการประชุมทางวิชาการนานาชาติ ตลอดจนขอขอบพระคุณสำนักงานคณะกรรมการการอุดมศึกษาและสำนักงานกองทุนสนับสนุนการวิจัยที่ให้การสนับสนุนทุนวิจัยในโครงการวิจัยนี้ สุดท้ายขอขอบพระคุณผู้บริหารมหาวิทยาลัยกรุงเทพ ที่ให้ความอนุเคราะห์สนับสนุนเรื่องเวลาการทำวิจัยและค่าใช้จ่ายในการเดินทางไปเสนอผลงานการวิจัยต่อที่ประชุมทางวิชาการทั้งในประเทศและต่างประเทศตลอดมา

ผศ.ดร.ปณิธิ เนตินันทน์

30 ส.ค. 2549

บทคัดย่อ

รหัสโครงการ MRG4780168

ชื่อโครงการ การสนับสนุนการออกแบบระบบปฏิบัติการที่สามารถขยาย และยืดหยุ่นได้ด้วยโครงร่างเชิงลักษณะ

ชื่อนักวิจัย ผู้ช่วยศาสตราจารย์ ดร.ปณิธิ เนตินันทน์
มหาวิทยาลัยกรุงเทพ

E-mail Address: paniti.n@bu.ac.th

ระยะเวลาโครงการ 1 กรกฎาคม 2547 – 30 มิถุนายน 2549

ในการพัฒนาระบบซอฟแวร์ เช่น ระบบปฏิบัติการ การมีปฏิกิริยากับส่วนประกอบต่าง ๆ ทำให้มีซับซ้อนมาก และยังเป็นข้อจำกัดให้การนำกลับมาใช้ การปรับแต่งระบบ ตลอดจนตรวจสอบการออกแบบและความถูกต้องของระบบเป็นไปด้วยความยากลำบาก ส่งผลให้ความต้องการใหม่ ๆ ที่จะเพิ่มลงไปในระบบไม่สามารถหลีกเหลี่ยงการออกแบบระบบใหม่หมด เป็นความเข้าใจผิดที่ว่าการนำกลับมาใช้ การปรับแต่งระบบ ตลอดจนตรวจสอบการออกแบบและความถูกต้องของระบบนั้นไม่จำเป็น แต่ซอฟแวร์ระบบต้องถูกออกแบบให้มีความสามารถโดยเฉพาะการนำกลับมาใช้ การขยาย และการปรับแต่ง อย่างไรก็ตามการสนับสนุนในเรื่องดังกล่าวเป็นเรื่องยากที่สามารถทำให้สำเร็จได้โดยใช้หลักการของการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) การเขียนโปรแกรมเชิงโครงข่าย (Aspect-Oriented Programming) เป็นอย่างหนึ่งที่เสนอขึ้นเพื่อมุ่งที่การแยกส่วนประกอบและลักษณะต่างๆ ในซอฟแวร์ออกจากกันตั้งแต่การเริ่มต้นของวงจรการออกแบบซอฟแวร์แล้วทำการรวมส่วนประกอบและลักษณะต่างๆ เข้าด้วยกันในขั้นตอนของการดำเนินการสร้าง นอกจากนั้นการเขียนโปรแกรมเชิงโครงข่ายสนับสนุนการแยกโครงข่ายต่าง ๆ ในส่วนประกอบของซอฟแวร์ได้อย่างเป็นธรรมชาติ แต่กระนั้นก็ดีวิศวกรรมซอฟแวร์โครงร่างเชิงลักษณะสามารถถูกสนับสนุนได้เป็นอย่างดีหากมีระบบปฏิบัติการที่ถูกสร้างขึ้นบนพื้นฐานของการออกแบบเชิงลักษณะ โครงการวิจัยนี้แสดงให้เห็นความเป็นไปได้ในการใช้โครงร่างเชิงลักษณะช่วยให้การออกแบบระบบสามารถเข้าใจได้ง่าย เพื่อสนับสนุนในการออกแบบระบบปฏิบัติการที่สามารถยืดหยุ่นและขยายได้ โครงร่างเชิงลักษณะใช้หลักสามมิติในการออกแบบ ประกอบด้วยส่วนประกอบย่อย (component) คุณลักษณะ (aspect) และระดับชั้น (layer) ซึ่งกระบวนการนี้สามารถสนับสนุนการนำกลับมาใช้ การปรับแต่ง และการยืดหยุ่นขยาย

คำหลัก ยืดหยุ่นได้ ขยายได้ โครงร่าง โครงร่างเชิงลักษณะ วิศวกรรมซอฟแวร์

Abstract

Project Code: MRG4780168

Project Title: Supporting the Design of Extensible and Adaptable Operating System Using Aspect-Oriented Framework

Investigator: Assistant Professor Dr.Paniti Netinant

Bangkok University

E-mail Address: paniti.n@bu.ac.th

Project Period: 1 July 2004 – 30 June 2006

With software systems such as operating systems, the interaction of their components becomes more complex. This interaction may limit reusability, adaptability, and make it difficult to validate the design and correctness of the system. As a result, re-engineering of these systems might be inevitable to meet future requirements. There is a general feeling that OOP promotes reuse and expandability by its very nature. This is a misconception as none of these issues is enforced. Rather, system software must be specifically designed for reuse, expandability, and adaptability. However, such support is difficult to accomplish using object-oriented programming (OOP). Aspect-Oriented Programming (AOP) is a paradigm proposal that aims at separating components and aspects from the early stages of the software life cycle, and combines them together at the implementation phase. Besides, Aspect- Oriented Programming promotes the separation of the different aspects of components in the system into their natural form. However, Aspect-Oriented software engineering can be supported well if there is an operating system, which is built based on an aspect- oriented design. This research will show an Aspect-Oriented Framework which simplifies system design by expressing its design at a higher level of abstraction, for supporting the design of adaptable and extensible operating systems. Aspect-Oriented Framework is based on a three-dimensional design that consists of components, aspects, and layers. This approach can support reusability, adaptability, and extensibility.

Keywords: Adaptability, Extensibility, Framework, Aspect Orientation, Software Engineering

ผลลัพธ์ (Output) โครงการที่ได้รับทุนจาก สกอ. และ สกว.

1. ผลงานตีพิมพ์ในวารสารวิชาการนานาชาติ

   อยู่ในระหว่างการพิจารณาของ International Journal of Software Engineering and Knowledge Engineering. Skokie, USA.

2. การนำผลงานวิจัยไปใช้ประโยชน์

   -เชิงสาธารณะ

   การวิจัยได้รับความสนใจในระดับนานาชาติ มีการจัดตั้งโครงการร่วมมือในการทำการวิจัยเรื่องอื่น ๆ ที่เกี่ยวข้องระหว่าง Bangkok University และสถาบัน Illinois Institute of Technology, Concurrent Programming research Group ประเทศสหรัฐอเมริกา

   -เชิงวิชาการ

   ได้รับเชิญให้เป็นประธานกลุ่มในการประชุมทางวิชาการนานาชาติ ณ ประเทศสหรัฐอเมริกา หัวข้อ Aspect-Orientation เมื่อเดือนมิถุนายน 2549 นอกจากนั้นยังได้นำความรู้ ทักษะ และผลการวิจัยที่ได้ไปใช้ในการเรียนการสอนวิชา Operating Systems ให้กับนักศึกษาระดับปริญญาตรี สาขาวิชาวิทยาการคอมพิวเตอร์

3. บทความที่ได้รับการตีพิมพ์เผยแพร่ในการประชุมทางวิชาการนานาชาติจำนวน 6 บทความ

   - Paniti Netinant and Tzilla Elrad. "A Framework for Extensible and Adaptable System Software" in *Proceedings of the International Conference on Programming Languages and Compilers (PLC 2005)*, Las Vegas, Nevada, USA, June 2005.
   - Paniti Netinant. "Component + Aspect = an Extensible and Adaptable System Software" in *Proceedings of the International Conference on Software Engineering Research and Practices(SERP 2005)*, Las Vegas, Nevada, USA, June 2005.
   - Paniti Netinant. "Extensibility Aspect-Oriented Framework to Build Agent-Based System Software" in *Proceedings of the 15th International Conference on Software Engineering and Data Engineering (SEDE 2006)*, Los Angeles, California, USA, July 2006.
   - Paniti Netinant. "Extensible and Adaptable System Software" in *Proceedings of the International Conference on Programming Languages and Compilers (PLC 2006)*, Las Vegas, Nevada, USA, June 2006.
   - Paniti Netinant. "Supporting Separation of Concerns to Automation of Code Generation" in *Proceedings of the International Conference on Software Engineering Research and Practices (SERP 2006)*, Las Vegas, Nevada, USA, June 2006.
   - Paniti Netinant. "Building Agent-Based System Software Using Aspect-Oriented Framework" in *Proceedings of the 2006 Electrical Engineering/Electronics, Computer, Telecommunication, and Information Technology (ECTI) International Conference*, Thailand, May 2006.

TABLE OF CONTENTS

Page

LIST OF FIGURES iii

LIST OF ABBREVIATIONS iv

Chapter

I. INTRODUCTION 1

Separation of Concerns 1
Criteria for Decomposition 2
Cohesion and Coupling 3
Advanced Separation of Concerns 3
Organization Theory 4
Research Objectives 5
Outline 6

II. BACKGROUND 7

Reflection and Metaobjects 7
Procedural Reflection 7
Metaobjects 9
Advanced Separation of Concerns 11
A Survey of Some Concerns and Their Separation 11
Problems with Scattered Code 15
Aspect-Oriented Programming 17
Other Work in Aspect-Oriented Software Development (AOSD) 22
Generative Programming 27
Intention Programming 30
Frameworks 31
Summary 32

III. THE FRAMEWORK 33

Architecture of the moderator pattern 33
Extensibility and Adaptability 34
Design Hierarchy 35
Composition of Aspects 36
Example: The Conference Room Reservation System 37
Relation between Moderator and Open Implementation 39
Comparison with other work 39
Summary 40

IV. REVISITED FRAMEWORK 41

An Aspect-Oriented Framework for Operating Systems 42

Page

Implementing Aspect-Oriented Framework 44
Summary 46

IV. CONCLUSION 47

Strength of This Research 47
Comprehensibility 48
Adaptability 48
Scalability and Expansibility 48
Reusability 49

REFERENCES 50

APPENDICES

A. FRAMEWORK TEMPLATE 64

B. READERS/WRITERS PROBLEM USING OBJECT-ORIENTATION 71

C. READERS/WRITERS PROBLEM USING THE ASPECT-ORIENTED FRAMEWORK 79

REPRINT 6 PAPERS PUBLISHED IN INTERNATIONAL CONFERENCES 92

LIST OF FIGURES

Figure 2.1: A Trigger for Logging Salary Increases.....12
Figure 2.2: A Cascading Style sheet Example..... 13
Figure 2.3: Separation of Concerns in WEB..... 14
Figure 2.4: Crosscutting Concerns..... 16
Figure 2.5: A Pictorial Representation of Crosscutting.....16
Figure 2.6: The Weaving Process..... 22
Figure 2.7: A Simple UML Tool Model Specification..... 24
Figure 2.8: Traversal/Visitor Specifications..... 24
Figure 2.9: Architecture for Event-based Dispatching..... 32
Figure 3.1. The Aspect Moderator..... 35
Figure 3.2. Design Hierarchy..... 36
Figure 3.3. The Moderator Interface..... 37
Figure 3.4. Implementation of the Aspect bank..... 37
Figure 3.5. Implementation of the room reservation system class..... 38
Figure 3.6. Implementation of pre-activation..... 38
Figure 3.7. Extensibility Aspect bank.....39
Figure 3.8. Comparison of the Framework.....39
Figure 4.1. Intra-Dependency..... 41
Figure 4.2. Inter-Dependency..... 41
Figure 4.3. PointCut Defines Inter-dependency..... 43
Figure 4.4. PointCut Defines Intra-dependency..... 43

LIST OF ABBREVIATIONS

ACS – Adaptive Computing Systems
AO – Aspect Oriented
AOD – Aspect-Oriented Design
AODSM – Aspect-Oriented Domain-Specific Modeling
AOP – Aspect-Oriented Programming
AOSD – Aspect-Oriented Software Development
AP – Adaptive Programming
API – Application Program Interface
ASDL – Abstract Syntax Description Language
ASOC – Advanced Separation of Concerns
AST – Abstract Syntax Tree
ATR – Automatic Target Recognition
C3I – Command, Control, Communication, and Information
CCM – CORBA Component Model
CDL – Contract Description Language
CF – Composition Filters
CLOS – Common Lisp Object System
CORBA – Common Object Request Broker Architecture
CSS – Cascading Style Sheet
DLL – Dynamic Link Library
DOC – Distributed Object Computing
DOM – Document Object Model
DSL – Domain-Specific Language
DSM – Domain-Specific Modeling
DSVL – Domain-Specific Visual Language
DTD – Document Type Definition
ECBS – Engineering of Computer-Based Systems
ECL – Embedded Constraint Language
ECOOP – European Conference on Object-Oriented Programming
GME – Generic Model Editor
GP – Generative Programming
GUI – Graphical User Interface
ICSE – International Conference on Software Engineering
IDE – Integrated Development Environment
IP – Intentional programming
ISIS – Institute for Software Integrated Systems
JSP – JavaServer Pages
JTS – Jakarta Tool Suite
KWIC – Key Word in Context
MCL – Multigraph Constraint Language
MDA – Model-Driven Architecture
MDSOC – Multi-Dimensional Separation of Concerns
MIC – Model-Integrated Computing
MOBIES – Model-Based Integration of Embedded Software
MOP – Metaobject Protocol
NCI – National Compiler Infrastructure
OCL – Object Constraint Language
OMG – Object Management Group

OO – Object Oriented
OOP – Object-Oriented Programming
OOPSLA – Object-Oriented Programming, Systems, Languages, and Applications
PCCTS – Purdue Compiler Construction Tool
PCES – Program Composition for Embedded Systems
QoS – Quality of Service
SOP – Subject-Oriented Programming
StratGen – Strategy Code Generator
SUIF – Stanford University Intermediate Format
UAV – Unmanned Aerial Vehicle
UML – Unified Modeling Language
WCET – Worst Case Execution Time
XML – Extensible Markup Language
XSLT – Extensible Stylesheet Transformations
YACC – Yet Another Compiler-Compiler

CHAPTER I

INTRODUCTION

In any engineering endeavor, a key requirement is the ability to compose large structures from a set of primitive elements. This is true for children who are constructing toy models of bridges and buildings using Lego" or Erector" sets. This is true, on a larger scale, for civil engineers who design and supervise the construction of skyscrapers.

This is especially true for software engineers who compose increasingly complex systems from components, classes, and methods. An important difference between the engineering of software, and the other undertakings enumerated above, is the recognition that the set of available core elements for software construction is often significantly larger. The composition of these elements can be specified at a much finer level of granularity. As a contrast, the bricks used to build Lego" houses, or the steel beams used in the construction of a bridge, come in but a few different shapes and sizes, and are composed using a simple standard interface (e.g., the prong and receptacle parts of a Lego" block have been unchanged since 1932 [Lego, 2002]; likewise, since around 1850, the standard dimensions for an air cell masonry brick in the United States has been 2.5 x 3.75 x 8 inches [Chrysler and Escobar, 2000]).

Separation of Concerns

Furthermore, the compositional permutations and dynamic interactions that are possible with software elements are several orders of magnitude richer than those found in other engineering activities. For example, a generic function can be parameterized with a seemingly unlimited number of other elements (e.g., a template function that can sort any data type using numerous factors). Parametric polymorphism is but one factor that contributes to the exponential state explosion problem that makes the composition of software so difficult. A reason for this complexity is that the essence of software elements is expressed as logical abstractions, as opposed to physical materials, which results in the generation of an enormous state-space that must be tested. In fact, the core of Brooks No Silver Bullet essay is a commentary that the molding of complex conceptual entities is the essence of software construction [Brooks, 1995].

It has been a longstanding understanding among software engineering researchers that the proverbial Gordian knot has appeared as a consequence of the exponential complexities involved in composing a set of software building blocks, or modules. Separation of concerns has emerged at the center of many helpful techniques for loosening the grip of this knot. Separation of concerns is not a new idea. In fact, over the past quarter-century, issues related to concern separation have been at the heart of the intersection of software engineering and programming language design research. A *concern* is generally defined as some piece of a problem whose isolation as a unique conceptual unit results in a desirable property. Concerns arise as intentional artifacts of a system. They are the primary stimulus for structuring software into localized modules.

The *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* defines a concern as, those interests that pertain to the systems development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders. Concerns include system considerations such as performance, reliability, security, distribution, and evolvability [IEEE 1471, 2000]. Other researchers have defined a concern to be, any matter of interest in a software system [Sutton and Rouvellou, 2001], and

a slice through the problem domain that addresses a single issue [Nelson et al., 2001]. Concerns are a central point of interest at any stage of the development cycle. A criterion for decomposition abstraction is doing just what our small minds need: making it possible for us to think about important properties of our program its behavior without having to think about the entirety of the machinations [Kiczales, 1992].

Modularity, abstraction, information hiding, and variability are important topics in software engineering that are associated with separation of concerns [Schach, 2002]. A clean separation of concerns provides a system developer with more coherent and manageable modules. From the structured paradigm of the 1960s and 1970s, to the Object-Oriented (OO) paradigm of the past few decades, there has always been an interest in creating new abstraction mechanisms that provide improved separation of concerns. There are several new paradigms on the horizon, as will be discussed in the next subsection (Advanced Separation of Concerns), to assist in further separation.

The most influential paper related to the study of modularization, and perhaps even in all of software engineering, is David Parnas on the Criteria to Be Used in Decomposing Systems into Modules [Parnas, 1972]. Parnas criteria aid a designer in achieving module independence. Parnas recognized that the decomposition of a system into its constituent parts must be performed with several specific goals in mind. To illustrate the consequences and tradeoffs from different design decisions, Parnas introduced a simple indexing program called KWIC (Key Word in Context). From a comparison of two separate modularizations for KWIC, Parnas suggested that modules be composed with the following objectives: changeability, independent development, and comprehensibility. The criterion of information hiding was shown by Parnas to be important in all three of these objectives.

Criteria for Decomposition

Changeability is the way to evaluate a modular decomposition, particularly one that claims to rest on information hiding, is to ask what changes it accommodates [Hoffman and Weiss, 2001].

A change to a module should not necessitate numerous invasive changes to many other modules. Parnas work has revealed that the structure of a system has a direct effect on the cost of change and maintenance. The potential that a module will undergo change should always be kept in mind when considering several different possibilities for modularization. Those implementation decisions that have the possibility of being changed, or those decisions that offer the most degree of flexibility in adaptation, should be hidden from the client of that module. This observation was a key toward the discovery of the properties of encapsulation and information hiding, where abstraction is the principal idea for delimiting what from the how. Designs that are created with the principle of information hiding permit the substitution of different implementations for the same abstraction. This improves the capacity to make changes based upon different desiderata (e.g., the typical time versus space arguments in data structure implementation).

Modularity is about separation: When we worry about a small set of related things, we locate them in the same place. This is how thousands of programmers can work on the same source code and make progress [Gabriel and Goldman, 2000].

As the complexity and size of software system soars, the ability of developers to independently work on separate modules becomes increasingly important. This is a vital attribute of the open-source community, where multiple developers work independently on a common collection of source code. The task of modularization, then, turns out to be a type of work assignment for each developer. The details of the design decisions and responsibilities of each developer should be hidden behind an exposed abstract interface. The interface

supplies the only means of access to the services offered by the module.

Comprehensibility in many pieces of code the problem of disorientation is acute. People have no idea what each component of the code is for and they experience considerable mental stress as a result [Gabriel, 1995]. When Microsoft first began conducting usability studies in the late 1980s to figure out how to make their products easier to use, their researchers found that 6 to 8 out of 10 users couldn't understand the user interface and get to most of the features [Maguire, 1994].

Comprehensibility can be negatively affected, within any context, by a poorly designed interface. Comprehensibility is a major goal of modular reasoning; that is, it should be possible for a developer to study one module at a time without being overwhelmed with the details of extraneous implementation information defined outside of the module context. Several popular ideas in software engineering (e.g., Dijkstra's Go To Statement Considered Harmful [Dijkstra, 1968], and Wulf and Shaws Global Variables Considered Harmful [Wulf and Shaw, 1973]), were in fact arguments made from the perspective of comprehensibility. An early result of object-oriented research demonstrated a strong link between comprehensibility and low coupling [Lieberherr and Holland, 1989].

Cohesion and Coupling

Cohesion and coupling are an obvious connection exists between highly cohesive and lowly coupled modules, and the objectives identified by Parnas. The seminal definitions of cohesion and coupling were provided within the context of structured design [Stevens et al., 1974]. A measure of cohesion and coupling can often provide an assessment of the quality of a design. Cohesion represents the degree of functional correlation between the individual pieces of a module (i.e., the extent to which a module is concentrated on a specific, well-defined concept). A method that exhibits low cohesion often contains code to perform several tasks that are conceptually different (e.g., a stack class where the push method also computes a square root). In a highly cohesive module, the various relationships within the module can be easily discerned because of the distinct focus of the module. This is a great attribute for supporting independent development.

Coupling can be described as the extent to which modules are connected with each other. Highly coupled modules are very brittle because a change to one module often requires the modification of a number of other modules. This also negatively affects independent development because highly coupled modules will often reveal their underlying internal implementation details to other modules. The comprehensibility of such modules is reduced, too, because several different modules must be examined to understand the intent of a module. Coupling is, to a large extent, the opposite of good modularity.

Advanced Separation of Concerns

Even though the general notion of separation of concerns is an old idea, one can witness the nascence of a research area devoted to the investigation of new techniques to support advanced separation of concerns. Recall that the opening paragraphs of this chapter highlighted the importance of modular composition within several engineering activities. It has been recognized by numerous researchers that the software modularization constructs developed over the past quarter-century are sometimes inadequate for capturing certain types of concerns. This has serious consequences with respect to modular composition.

Previously defined modularization constructs are most beneficial at separating concerns that are orthogonal [Tarr et al., 1999]. However, these constructs often fail to capture the isolation of concerns that are non-orthogonal. Such concerns are said to be

crosscutting, and their representation is scattered across the description of numerous other concerns. Crosscutting concerns are denigrated to second-class citizens in most languages (i.e., there is no explicit representation for modularization of crosscutting concerns). As a result, crosscutting concerns are difficult to compose and change without invasively modifying the description of other concerns (i.e., crosscuts are highly coupled with other concerns). The three objectives of changeability, independent development, and comprehensibility are sacrificed in the presence of crosscutting concerns because of the lack of support for modularization (see [Gudmundson and Kiczales, 2001] for an evaluation of these objectives in the context of newly proposed modularization constructs). The latest research efforts, under the general name of Aspect-Oriented Software Development (AOSD) [AOSD, 2002], explore fundamentally new ways to carve a system into a set of elemental parts in order to support crosscutting concerns. The goal is to capture crosscuts in a modular way with new language constructs called *aspects*. A large portion of the second chapter thoroughly explains the problem of crosscutting concerns and surveys solution techniques.

The next section is not about AOSD, but rather shows how crosscutting enters into other areas of human life, as well.

Organization Theory

Thus my central theme is that complexity frequently takes the form of hierarchy and that hierarchic systems have some common properties independent of their specific content [Simon, 1996]. Various types of organizations encompass elaborate hierarchies. The subject of organizational hierarchy has been studied for nearly a century. Within the disciplines of management and administration sciences, there is a popular corpus known as organization theory. Organization theory has a basis for comparison with software organizational hierarchy has been studied for nearly a century. Within the disciplines of management and administration sciences, there is a popular corpus known as organization theory. Organization theory has a basis for comparison with software development whenever a hierarchic approach to software decomposition is adopted. It is worth noting that some of the influential work in organization theory was conducted by a Turing Award winner Herbert Simon who also received the Nobel Prize for his work on decision-making in organizations. This section will offer a short assessment of organization theory as it relates to software construction

Since Adam Smiths, The Wealth of Nations [Smith, 1776], the concept of division of labor has been an important topic within the discourse of economics, and the study of supporting institutions. A keen contribution by Smith was a quantifiable justification for the benefits that division of labor and specialization garner vis-à-vis efficiency and productivity. Division of labor is to a large extent correlated to the general objectives of separation of concerns as it relates to information hiding and the independent development of modules. Parnas actually gave a definition for the term module that would support such an assertion, as he stated, in this context module is considered to be a responsibility assignment rather than a subprogram [Parnas, 1972]. The responsibility assignment of a module to a programmer relates to the specialization of effort that exists in division of labor. Interdependent Organizations display degrees of internal interdependence. Changes in one component or subpart of an organization frequently have repercussions for other parts the pieces are interconnected [Daft et al., 1987]. After an organization is hierarchically constructed (as a result of the specialization of labor division), it is almost assured that the boundaries of the hierarchy will be broken as a result of interdependence among the different divisions. Large organizations naturally have certain kinds of concerns that are non-orthogonal to the hierarchic structure. Such facets of the organization increase the coupling of each division of

the organization and expose particular characteristics of the division specialization (an example of this is provided in the next section, within the context of a student requesting a transcript). These are the crosscutting concerns of the organization. Studies have been conducted on the mechanisms by which organizations have the ability to adapt to feedback [Daft et al., 1987]. These self-correcting behaviors are analogous to the reflective methods that are surveyed in Chapter 2. Hierarchic decomposition is a tool for accomplishing goals and objectives within an organization. It is normal for organizations to have multiple goals, some of which may be conflicting [Hall, 1998]. The multiple rules that are spread throughout the hierarchy of an organization are the result, in many cases, of the implementation of some policy, or protocol. A policy is a mechanism that coordinates specific objectives across a set of dislocated organizational units. A policy, and the rules that implement it, could be considered a type of crosscutting concern within the organization. The pejorative meaning of red-tape is tied to the frustrations that result from bureaucratic rules of policy implementation. In order for the policy to be realized, the specialization of many different organizational departments is needed. Intriguingly, the initial concept of bureaucracy, as proposed by [Weber, 1946], was promoted as the best structure for dealing with a changing environment today, it is mostly associated with a negative connotation. An interesting case study is presented in [Perrow, 1986], where a formal process at the University of Wisconsin was scrutinized. The policy that was examined corresponded to the process for a university faculty member to make a formal suggestion, or complaint. It was discovered that a complete review of the formal request would require that it pass through over fifteen levels of the university hierarchy. This example is comparable to crosscutting concerns in software implementations that execute a protocol across a large code base. As will be shown in a later chapter (see Figure 9 through Figure 11), the communication path in a hierarchy can introduce unnecessary overhead in both organizations and software. The concept of an Independent Integrator has been advocated as a coordinator of the policies involving myriad interdependent departments [Dessler, 1986]. An integrator is the closest entity within organization theory that has a relation to techniques for advanced separation of concerns. The role of an integrator is to step outside the hierarchical bounds and assist in the weaving of a crosscutting policy throughout the organization.

Research Objectives

This research is about advanced separation of concerns at the system modeling level, and the construction of support methodology for system software that facilitate the elevation of crosscutting modeling concerns to first-class citizens (i.e., explicit constructs for the representation of such concerns) where adaptability and extensibility can be achieved. The contributions described in this research can be summarized by two research objectives: Raise Aspect-Oriented (AO) concepts for supporting the design of adaptable and extensible system software, such as operating systems, to a higher level of abstraction. An aspect orientation can be beneficial at different stages of the software lifecycle and at various levels of abstraction; that is, it also can be advantageous to apply aspect orientation at levels closer to the problem space (e.g., analysis, design, and modeling), as opposed to the solution space (e.g., implementation and coding). Whenever the description of a software artifact exhibits crosscutting structure, the principles of modularity espoused by aspect orientation offer a powerful technology for supporting better separation of concerns, which is ease of reuse, adaptability, extensibility, and comprehensibility. This has been found to be true also in the area of domain-specific modeling [Gray et al., 2000]. Although there have been other efforts that explore AO at the design and analysis levels (see Chapter 2 for more details), the work described in [Gray et al., 2001a] represents the first occurrence in the literature of an actual

aspect-oriented weaver (see Figure 2.6 in Chapter 2) that is focused on system modeling issues, rather than topics that are applicable to traditional programming languages.

The research assists in the creation of new weavers using a generative framework. Because the syntax and semantics of each modeling domain are unique, a different weaver is needed for each domain. These two objectives provide a contribution toward the synergy of AOSD and Model-Integrated Computing (MIC) (see [Sztipanovits and Karsai, 1997] for an overview of MIC). This union assists a modeler in capturing concerns that, heretofore, were very difficult, if not impossible, to modularize. A key benefit is the ability to explore numerous scenarios by considering crosscutting modeling concerns as aspects that can be rapidly inserted and removed from a model.

This research use the Aspect-Oriented Framework (AOF) developed by Netinant and Elrad to design an operating system built on separation of aspectual system properties from basic functionalities. We believe this is a solid break through and innovative approach to advance understanding and capabilities of system software development and utilization in operating system area. The project will investigate the potential of building Aspects and Components-Oriented Operating Systems (ACOOS) with respect to the following demands.

1. The impact of the aspect-oriented framework called component, adaptability, and layers (CAL) to support the design of operating systems on the extendibility and adaptability of current and potential new systems features.

2. The impact of potential use of aspect orientation approach to operating system design and implementation.

3. The impact of the design and implementation for the aspect and component-oriented operating systems on the ease of implementation and extensibility.

4. The impact of the design and implementation for the aspect and component-oriented operating systems on the ease of implementation and adaptability.

The goal of this two-year project is a development of an open architecture, a prototype of an aspect and component-oriented operating system called ACOOS using aspect-oriented frameworks (CAL) where both basic functional components and crosscutting system properties are designed separately from each other in each layer. Their composition is formally supported to ensure correctness. This separation of concerns allows for reusability and enables the building of software systems that are comprehensible, adaptable, and extendable.

Our research concentrates on the design of extensible and adaptable operating systems using aspect-oriented frameworks. We need to address the following two issues: what should be done in aspects and how it should be done. Based on the current state of the art using an aspect-oriented design framework

Outline

A background survey of related literature can be found in Chapter 2. The chapter reviews several techniques that have been used over the past decade to provide the variability needed to support clean separation of concerns. That chapters overview begins by examining topics such as reflection and metaprogramming. The Chapter 2 also provides the incentive for, and summary of, the emerging research efforts in advanced separation of concerns. Within the general context of generative programming, a cornucopia of topics is summarized at the end of the second chapter. This encompasses a brief synopsis of the literature on object-oriented frameworks, code generators, and domain-specific languages.

In Chapter 3, the framework is introduced. Chapter 4 is about concluding and remarks of the framework. Finally, Chapter 5 is conclusion of this research. A comprehensive bibliography is included at the end of this report.

CHAPTER II

BACKGROUND

This chapter contains a broad survey of many techniques that have been found useful for supporting modularization of software (e.g., reflection and metaobjects, advanced separation of concerns, generative programming, and frameworks). These techniques also are effective at providing the capability needed for software compositions to adapt and change to evolving requirements. The contributions of this research in Chapters 4 are extensions of several of these ideas.

Reflection and Metaobjects

Industry increasingly demands that systems be adaptable and extensible. This demand may be manifested in various forms, including:

- The malleability of an application with respect to a set of changing user requirements (i.e., the degree of difficulty to affect change in an application's source code implementation);
- The degree of adaptability within a system in the presence of a changing environment (i.e., the capacity of an application to examine itself and modify its own internal state during run-time).

Reflection and metaprogramming provide powerful techniques for extensibility by separating the program's computation (the base level) from the specifics of how the program is interpreted (the metalevel). This separation permits the modification of the underlying implementation semantics (through changes to the metalevel) at run-time. These techniques have been shown to provide great flexibility in systems that must adapt to changing environments [Robertson and Brady, 1999]. A philosophical definition of reflection has been given as, "...the capacity to represent our ideas and to make them the object of our own thoughts" [Clavel, 2000]. As used in this sense, reflection was first introduced in logic as a way to extend theories [Hoftstadter, 1979]. Reflection also has been an active research area within the context of programming languages. Various forms of reflection are even appearing in popular programming languages like Java.

Procedural Reflection

The work of Brian Cantwell Smith provided the seminal ideas for formally applying reflection to programming languages [Smith, 1982]. Smith defined procedural reflection as the concept of a program knowing about its implementation and the context in which it is executed (later, Smith would prefer the term introspection in place of procedural reflection). A reflective system is capable of reasoning about itself in the same way that it can reason about the state of some part of the external world. Introspection offers the capability of dynamically adjusting the way that programs are executed. A reflective system has a causally connected self-representation [Smith, 1982]. Thus, a reflective system has access to the structures that are used to represent it. Depending on the level of support for reflection, these internal representations can be inspected and even manipulated. Here, the term "causally connected" means that a manipulation of the internal representation structures directly affects the observable external behavior.

Smith identified three conditions that must be satisfied in order for a system to be considered introspective:

1. The system must be able to represent a description of its internal structure in such a way that it can be inspected and modified by facilities within the system.

2. The self-representation must be causally connected to the structure and behavior of the system. Each event and state in the system must be self-described and modifications to the description must result in a change in structure or behavior.

3. The self-representation must be at the proper level of abstraction. It must be low enough such that meaningful modifications can be made. Yet, it must not be so low-level that a programmer gets bogged down in a morass of detail.

Metacircular Interpreters

Smith also described a language, called 3-Lisp that supported his model of reflection. In 3-Lisp, the notion of a reflective tower of metacircular interpreters [Steele and Sussman, 1978] supports the incremental changes to layers of interpreters. A *metacircular* interpreter is a program that is written in the same language that it interprets [Abelson and Sussman, 1996]. The reflective tower is an infinitely ascending stack of interpreters. All interpreters in this tower are implemented in 3-Lisp. Each new layer in the tower is interpreted by the layer above it. The interpreter at the very bottom of the layer is the traditional program that processes user input. In 3-Lisp, as is typical of most Lisp or Scheme implementations, an expression, an environment, and a continuation argument capture the state of an interpreter. The layers in the tower are connected by reification and reflection. Reification is the inverse of reflection – it is about the ability to consider an abstract concept as concrete. Sobel and

Friedman distinguish the two processes as, "...converting some component of the interpreter's state into a value that may be manipulated by the program is called *reification*; the process of converting a programmatically expressed value into a component of the interpreter's state is called *reflection*" [Sobel and Friedman, 1996]

Object Reflection

The first effort to incorporate "Smithsonian" reflection into an object-oriented language is described in [Maes, 1987]. Building on the foundation of procedural reflection, an object-oriented reflective architecture divides the object part from the reflective part. The object part describes and manipulates the application domain and the reflective part describes and manipulates the object computation semantics.

The reflective operations provided by some object-oriented programming languages are limited. For example, the model of reflection provided in Java is much weaker than that found in Smalltalk and the Common Lisp Object System (CLOS). The reflection mechanism in Java does not permit the modification of the internal representation [Anderson and Hickey, 1999], [Sullivan, 2001]. It only provides a type of "read-only" examination facility that allows run-time inspection of the internal representation of an object. A further limitation is that the reflective methods in Java are marked final, which prohibits their extension. Therefore, the reflective model provided in Java is not of the Smithsonian style because it does not provide the adaptation needed for being causally connected. The definition of introspection is presented slightly differently in [Bobrow et al., 1993]. They define *introspection* as a program's ability to observe and reason about its own state. They define *intercession* as the more powerful capability of modifying the internal state to affect the underlying semantics. Using these definitions, Java can be said to provide support for introspection, but not intercession.

Metaobjects

Meta means that you step back from your own place. What you used to do is now what you see. What you were is now what you act on. Verbs turn to nouns. What you used to think of as a pattern is now treated as a thing to put in the slot of another pattern. A metafoo is a foo into whose slots you can put parts of a foo [Steele, 1998]. As Steele observes, the prefix meta is used to denote a description that is one level higher than the standard frame of perception. Meta is also used to mean "about," "between," "over," or "after." Hence, a metaprogram is usually defined as a program that modifies or generates other programs. A compiler is an example of a metaprogram because it takes a program in one notation as input and produces another program (usually object code) as output. Reflection is considered a form of metaprogramming where the target of the modification is the metaprogram itself. Metaprogramming can be a complex activity sometimes because there can be a blur between the base level and the metalevel.

Metaobject Protocols

Maes appears to be the first to introduce the notion of a metaobject [Maes, 1987]. In an object reflection system, a metaobject is just like any other object during run-time. Every object in the language has a corresponding metaobject and every metaobject has a pointer to its corresponding implementation object [Maes, 1988]. The metaobject contains information about its language object, such as details on its implementation and interpretation. During the execution of a system, the language objects may request information about their state, and even perform a modification on the internal representation. Metaobject Protocols (MOPs) facilitate the modification of the semantics of the underlying implementation language [Kiczales et al., 1991]. Manipulating the interfaces that the MOP provides can incrementally modify the behavior and implementation of the underlying language. For example, CLOS has a MOP that specifies a set of generic functions [Steele, 1990].

There are five categories of functions that represent the core elements of CLOS (i.e., classes, slots, methods, generic functions, and method combination). A metaobject represents each of these core elements. Each metaobject has a metaclass. The metaclasses behave like any other class such that the semantics of a metaobject can be adapted by modifying its metaclass. A programmer can alter the semantics of CLOS by using standard object-oriented techniques, like subclassing. The instance of each metaobject can be adapted at run-time. The behavior of the system at any particular time is dependent on the configuration of the set of metaobjects. The protocol, in this case, represents the interfaces of the metaclasses. Any modification to the behavior of the system must adhere to the interface definitions. MOPs gain their adaptive power from a synergy of reflection and Object-Oriented Programming (OOP). As described in [Kiczales et al., 1991], there are three attributes of a metaobject protocol:

1. The core programming elements of a language are represented as objects. For example, the syntax and semantics for method calls, the rules for handling multiple-inheritance, and the rules of method lookup are all represented as objects.
2. The behavior of the language is encoded in a protocol based on these objects. The protocol is the interface of the metaclasses.
3. A default object is created for each kind of metaobject.

Concerning the first attribute from above, an example of the ability to modify multiple-inheritance rules is shown in [Kiczales et al., 1991]. A generic function called compute-class-precedence-list returns the rules that determine the resolution of conflicts due to multiple-

inheritance. The programmer can modify this list so that new rules of conflict resolution are used. As another example, objects are created in CLOS by calling make-instance. The implementation of this method can be redefined at runtime to perform specialized adaptations during object creation. Although the majority of the literature on reflection and metaprogramming is described in some dialect of Lisp, there have been efforts to apply these techniques to other languages. For example, [Chiba and Masuda, 1993] describe a basic metaobject protocol for a language called Open C++. A more detailed description of a MOP for C++ is given in [Forman and Danforth, 1999]. While not analogous to MOPs, *per se*, there has also been research in C++ on an idea called static metaprogramming. A variant of this, which relies on C++ templates, provides a compile-time facility for generating code and component configuration [Czarnecki and Eisenecker, 2000].

Metaobjects also can be used in assisting in the separation of concerns in areas other than programming languages. Research at IBM recognized that, within middleware, there is an intermixing of application code and protocol code [Atsley et al., 2001]. The lack of modularity affects the ability to maintain and customize the middleware. A metaobject protocol cleanly separates the policy and protocol code from the underlying application. Some example metaobjects that were defined to represent communication events are transmit (what happens when a component sends a message), deliver (what happens when a message is received by a component), and dispatch (the received message a component decides to process). Nonfunctional system properties like security and persistence [Rashid, 2002] can be cleanly separated from the base level program to improve reuse. This has been termed *implementational reflection* in [Rao, 1991].

Within the scope of distributed object computing and middleware, the technique of CORBA interceptors is closely related to metaobject protocols. Interceptors are defined as, "non-application components that can alter application behavior" [Narasimhan et al., 1999]. An interceptor can transparently modify the behavior of an application by attaching itself to the invocation path of a client and server object. Interceptors have been shown to be useful in enhancing CORBA by providing adaptability with respect to profiling, protocol adaptation, scheduling, and fault tolerance [Narasimhan et al., 1999].

Evaluating MOPs

A detailed evaluation of the practical use of MOPs can be found in [Lee and Zachary, 1995]. In this study, a MOP was applied to a geometric CAD tool in order to add persistence to the CLOS implementation objects. The project was described as being very ambitious and a much more complicated application of MOPs than previously studied. Much of the evaluation was positive. Because the majority of the effort to extend CLOS related to objects, the metaobject protocol provided a useful resource. However, the effort had several difficulties. Although the CLOS MOP is very useful when extension is based on a property of an object, the protocol is not helpful when there is a requirement to augment a feature that is not captured as an object property. For example, in CLOS, arrays and several other composite values are native to Common Lisp and are not available for extension in the MOP. Another difficulty was found with respect to performance. In several experiments, it was found that object creation was sixteen times slower than the prior implementation that did not use a MOP. Similarly, write access using the MOP was found to be about seven times slower. Performance has always been a problem for reflective approaches. Consider the following observation, with respect to Java-based reflection, "As of release 1.4, reflective method invocation was forty times slower on my machine than normal method invocation. Reflection was re-architected in release 5 for greatly improved performance, but is still twice as slow as normal access, and the gap is unlikely to narrow" [Bloch, 2001]. The performance penalty

resulting from many dynamic calls in a reflective implementation will often rule-out reflection as an implementation alternative in some contexts.

Open Implementations

Traditionally, black-box abstraction states that a software module should expose its interface, but hide its implementation details. This is a corollary to [Parnas, 1972], and is similar to the *Open-Closed Principle*, described in [Meyer, 1997], which states that a module should be open for extension, yet closed for modification. However, the idea of an open implementation disagrees with this principle when applied fundamentally. Research in the area of open implementations has found that, in some cases, software can be more reusable when a client is allowed to control a module's implementation strategy [Kiczales, 1996]. Open implementation proponents agree that the base level should remain closed like a black-box. It is the metapart that they advocate opening to extension [Kiczales, 1992]. In fact, the initial motivation behind MOPs was a desire to open the language in such a way that better control could be exerted over the selection of the implementation with respect to certain performance concerns [Kiczales et al., 1993].

Advanced Separation of Concerns

In Chapter 1, the importance of separation of concerns was motivated. During the latter part of the 1990s, research in this area increased with an invigorated interest. This was due, in part, to the recognition that the languages and tools used to develop software hampered the proper isolation of specific categories of concerns. The inadequacies of modern programming languages (with respect to separating certain concerns) prompted many researchers to take a fresh look at modularization constructs and extensions/complements to current languages. The focus of the problem can be discerned from the observation that programming languages are often used in a linear process. However, the things that we want to express in a language, and our conceptualization of key abstractions as a supporting mechanism, are certainly not linear. This section provides the initial motivation and problems that are being solved by a new area of research entitled Advanced Separation of Concerns (ASOC).

A Survey of Some Concerns and Their Separation Before initiating the impetus behind advanced separation of concerns at the implementation level, it may be beneficial to first notice the various methods that have been suggested for managing concerns in other contexts. The examples in this section represent concerns that are typically identified outside of the milieu of traditional programming language research.

Database Triggers

Assume that the following business rule is to be consistently enforced within a database: "Every time an employee's salary is increased by 25%, log the employee's social-security number, previous salary, and new salary into an audit table." The implementation of this business rule requires that some action be taken every time that an update to the salary column occurs. This business rule is an archetype for a crosscutting concern. Without triggers, the realization of this rule would require that the concern be placed in *all* of the stored procedures that update the employee's salary. That is, the delta of a salary increase must be computed for each update and checked against the specified 25% rate increase. This could result in the insertion of redundant code throughout all stored procedures that are affected by this business rule. The problem is compounded when the salary update occurs

within embedded SQL in a base programming language. In that case, the check must be made outside of the database in every location of the base program that implements this business rule. Fortunately, a trigger mechanism facilitates a cleaner solution. A trigger-based solution, like that found in Figure 1, would provide a single location from which changes could be made to the semantics of the concern. The trigger solution does not need access to metalevel control in order to capture the intent of the concern (i.e., it is not necessary to redefine the underlying semantics of the table update definition). As will be shown later, this is similar to the way that AspectJ captures a concern without resorting to metaprogramming techniques (i.e., aspects and non-aspects are all at base-level code – there is no reference to the metalevel within AspectJ). This is an important point in differentiating triggers, and even aspect languages, from pure metaprogramming techniques. Later in this chapter, the constitutive parts of an aspect language will be described. A preview of these is now given in a comparison of aspect languages and triggers.

```
CREATE OR REPLACE TRIGGER salary_audit
AFTER UPDATE OF salary ON employee
FOR EACH ROW
WHEN (new.salary > 1.25 * old.salary)
CALL log_salary_audit(:new.ssn, :old.salary, :new.salary);
```

Figure 2.1: A Trigger for Logging Salary Increases

On the second line of Figure 1, the "AFTER UPDATE" statement indicates the point of execution when the trigger statement is applied. Using BEFORE/AFTER, an Oracle database trigger is able to influence the dynamic execution of a database server whenever certain operations (DELETE, INSERT, UPDATE) are executed on a database table. There are six different variations that can be given, resulting from the permutation of {BEFORE, AFTER} × {DELETE, INSERT, UPDATE}. Also, on the second line, the "OF salary ON employee" is similar to the pointcut idea in aspect languages. This construct identifies a particular point in the database table (e.g., a row and a table) that is affected by the trigger. The "when condition" syntactical construct on line 4 has some likeness to the "if" pointcut designator in AspectJ. The executable statement that is associated with the trigger (this is the action that occurs when the trigger is fired), found on the last line of Figure 1, is akin to the concept of "advice" in AspectJ. The definition of these aspect-oriented terms will be clarified in a subsequent section. Even though the database trigger mechanism permits the capture of crosscutting business rules within a database, it has several weaknesses when compared to pure aspect languages. The most evident limitation is the lack of the ability to create compositions of triggers. The trigger approach allows only the naming of a single table. It does not permit the logical composition of table property descriptions. That is, the type of pointcut model used within triggers is not composable in the same way as AspectJ. Triggers also do not support the concept of wildcards within the naming of a pointcut. For example, the second line from above could not be written as "OF sal* ON emp*" in order to designate multiple columns and tables that are affected by the trigger.

Mail Merge

Mail merge is an office automation tool that supports the separation of the form of a document from a data source of merge fields. By this separation, the insertion of each instance throughout the document can be better managed (see Figure 2). Consider the task of a lawyer who specializes in commercial foreclosures. He, or she, will typically need to process fifteen different documents in order to execute a foreclosure (according to

information obtained from a personal conversation with a Nashville attorney). Furthermore, five or more different parties (with separate contact information) are typically involved. Their contact addresses, and other pertinent information, are diffused across the space of the various legal documents. By separating the instance from the form, the author of the document is spared from the tedious task of visiting multiple locations in the document in order to make each change. Although the mail merge tool assists in a specific type of concern separation, it requires the document designer initially to visit every instantiation point in order to insert a field designator (because of this, the process is somewhat similar to the LaTeX macro command).

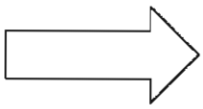
Style Sheets

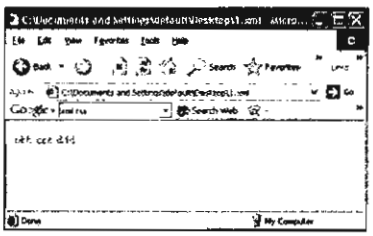
Within the context of web publishing, style sheets are a useful technique for separating the content of a document from its presentation style [Meyer, 2000]. Such a separation provides a method for making seamless global changes to the appearance of a document without the need for visiting numerous individual locations in the document. In a Cascading Style Sheet (CSS), a rendering engine visits each node of a document. As the traversal proceeds over the document's hierarchy, the rendered attempts to match the current element with a pattern specified as a CSS rule. A CSS rule consists of two parts: a selector, which names the type of the element to which the style will be applied, and a declaration, which represents the type of style to be applied.

XML Text

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="style1.css" type="text/css"?>

<FOO>
  <BAR1>bbb</BAR1>
  <BAR2>ccc
    <BAR3>ddd</BAR3>
  </BAR2>
</FOO>
```





CSS Stylesheet (style1.css)

```
BAR1  {color:red}
BAR2  {color:blue}
BAR3  {color:green}
```

Figure 2.2: A Cascading Stylesheet Example

An illustration of the application of a CSS rule is shown in Figure 2. The top-left of the figure contains the content of a document as represented in the Extensible Markup Language (XML). The information regarding the name of the specific style that is to be applied (in this case, the style sheet named style1.css) is located within the preamble of this document. The specification of style1.css is listed in the bottom-left of the figure. As can be seen, this style sheet has a rule asserting that all elements of type BAR1 are to be rendered in the color red. In this example, it should be understood that the rendering engine resides within the browser.

Literate Programming and WEB

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do [Knuth, 1984]. The idea of literate programming was initially described by Donald Knuth and implemented with a tool called WEB [Knuth, 1984]. In WEB, a single program is a combination of source code, documentation text, and WEB commands. Literate programming assists a programmer in assembling programs that are more easily read by a human. This is done by treating the construction of documentation and source code as a simultaneous activity. The aim is to make the construction of programs more like the creation of a literary work. The formal expression of a concern is so closely tied to the informal description that tools are needed to separate the two representations so that they are consumable by different parties (e.g., a compiler and a human). In WEB, source code is produced from the TANGLE tool, and documentation is formed by the WEAVE tool (see Figure 3). It is interesting to note that the structure of the process for creating WEB programs is almost opposite to that seen in Figure 1 and Figure 2. In those contexts, the concept of weaving a document entailed the notion of bringing separated entities together as one (where the separation provided some desirable property that assisted in change maintenance and comprehensibility). In literate programming, however, the concept of weaving represents the task of separating concerns of interest (e.g., the visual presentation of documentation) from an existing tightly coupled document.

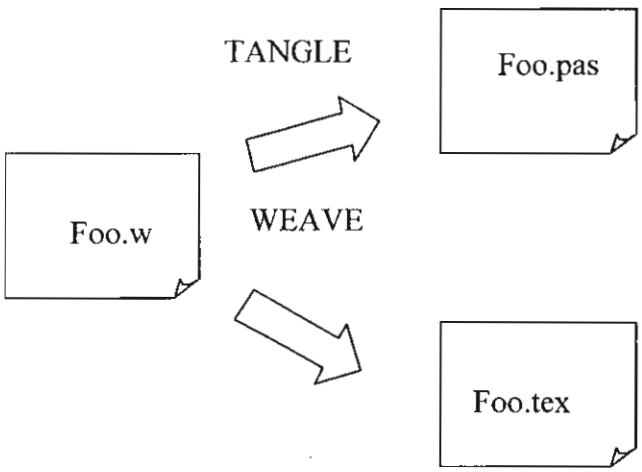


Figure 2.3: Separation of Concerns in WEB

The preceding subsections provided several examples of concern separation. Two of the four examples were in contexts not associated with software development (e.g., mail-merge and stylesheets). A common topic in each of these examples was the existence of an integration tool for assisting in the conceptual separation. In the following sections, the problems associated with crosscutting concerns are motivated, along with the need for a new type of software integration tool – a weaver.

Problems with Scattered Code

It is organization which gives birth to the dominion of the elected over the electors, of the mandataries over the mandators, of the delegates over the delegators. Who says organization, says oligarchy [Michels, 1915]. Non-orthogonal concerns can be described as crosscutting, because such concerns tend to be scattered across the traditional modularity boundaries provided by a development paradigm. In programming languages, two concerns crosscut when the modularity constructs of a language allow one concern to be captured separately, but only to the detriment of another concern that must be captured in a way that is not cleanly localized. This has been referred to as the "tyranny of the dominant decomposition" [Tarret al., 1999]. The "Iron Law of Oligarchy," quoted above from Michels, suggests that bureaucratic hierarchy tends to result in oligarchy; that is, those at the top of an organization are those that rule. In Chapter 1, an allusion was made to this tyranny under the Organization Theory section that described Interdependence. With respect to the dominant decomposition, this also seems to be true with traditional methods for software modularization. Crosscutting has the potential to destroy modularity. The crosscutting phenomenon can occur in structured programming, where the procedure, function, and module delimit the modularity boundaries. It is also prevalent in object-oriented programming, where classes, methods, and inheritance define the boundaries of encapsulation.

Persistence

Crosscutting concerns provide difficulties for a programmer because the implementation of the concern is scattered throughout the code; the concern is not localized in a single module. This can be a source of potential error when modifications are required. Comprehensibility is negatively affected in two ways [Tarr et al., 1999]:

- The scattering problem: The ability to reason about the effect of a concern is decreased because a programmer must visit numerous modular units in order to understand the intent of a single concern. The problem is that a concern often touches many different pieces of code.
- The tangling problem: Within a module, the tangling of numerous concerns decreases cohesion, and raises coupling. This reduces a programmer's ability to understand the core intent of a particular module. The problem is that many concerns may touch a single piece of code.

Programmers are often forced to keep track of crosscutting concerns in their heads. This is an error-prone activity, because even medium-sized programs can have hundreds of different crosscutting issues [Tristram, 2001]. Another problem of crosscutting concerns is maintenance. It is often the case that the global spreading of a concern, and the ramifications of its modifications, are not intuitive to those who inherit the code for maintenance. Maintenance becomes more of an archaeological metaphor, where a programmer must search through rubble in order to uncover a useful artifact [Hunt and Thomas, 2002]. The Parnasian objectives, found in Chapter 1, are usually sacrificed in the presence of non-orthogonal concerns.

Figure 4 provides an illustration of scattering and tangling. The three individual units (Unit A, B, and C) would be considered highly cohesive, if it were not for the tangling of the three concerns of logging, synchronization, and persistence. Furthermore, the scattering of these concerns would make it difficult to change their behavior, especially if the example were scaled to a much larger problem with thousands of units.

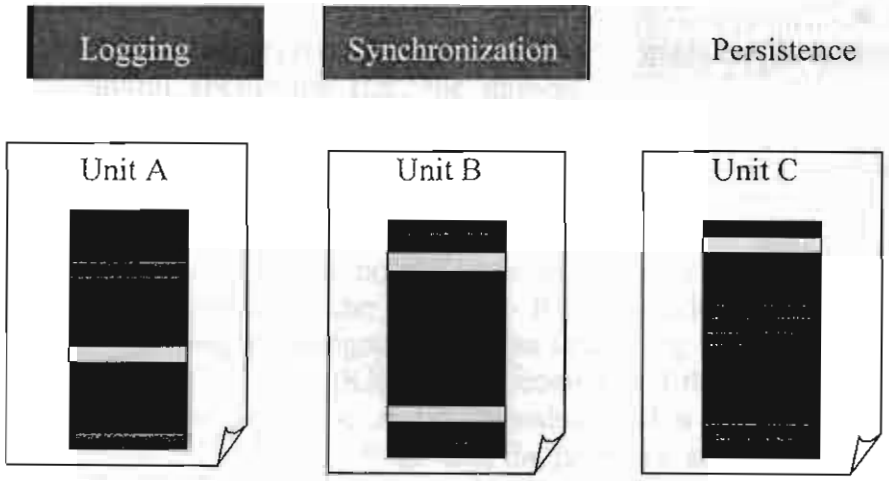


Figure 2.4: Crosscutting Concerns

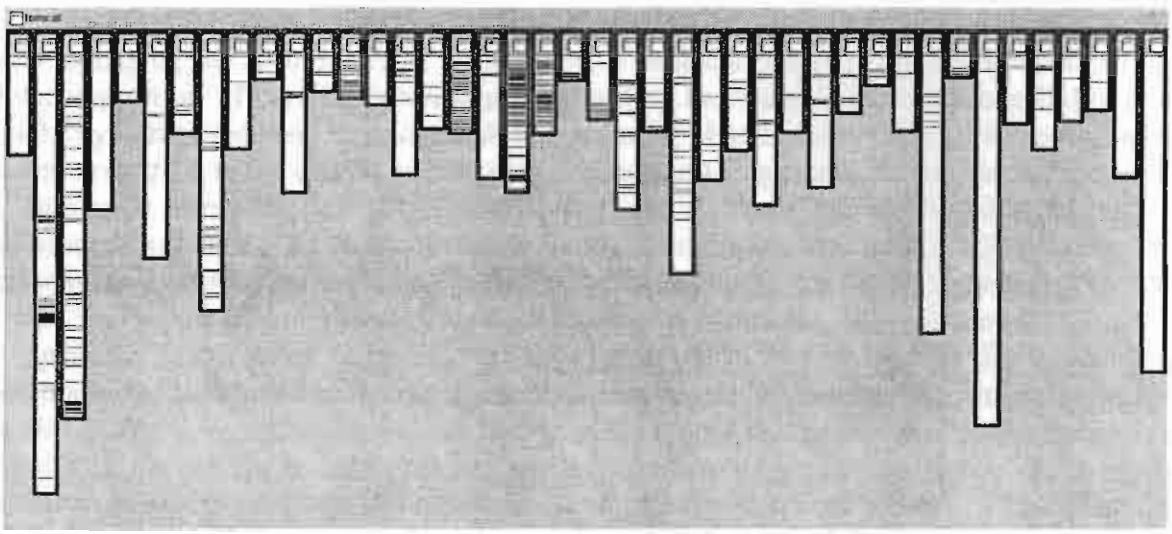


Figure 2.5: A Pictorial Representation of Crosscutting

This figure represents a piece of the Apache Tomcat code. Tomcat is an implementation of the Java Servlet and JavaServer Pages (JSP) specifications. Tomcat can run as a standalone, or it can be integrated into the Apache Web Server. The white vertical boxes represent a few of the classes in a subset of the Tomcat implementation. The highlighted lines designate the lines of code related to the concern of logging. Notice that the implementation of the logging concern is spread across the various classes. It is not located in a single spot. In fact, it is not even located in a small number of places. As reported in [Robillard and Murphy, 2002], a modification to the logging concern, "would require the developer to consider 47 of the 148 (32%) Java source files comprising the core of Tomcat." In this example, if the type of information to be logged is changed, then a developer may be required to make modifications to each of these 47 individual source files. From a software engineering viewpoint, this is not desirable. There is no cohesive module for representing the concept of logging – that concept is coupled among all of the other concerns. To highlight the importance of this, forget for a moment that the highlighted code in Figure 5 represents

logging. Assume, instead, that it represents all of the code for implementing the concerns of an employee in a payroll application (i.e., the implementation of employee features is scattered across multiple source files, in different modules). In that situation, it is easy to see that the basic principles of cohesion and coupling are being violated. The same can be said, then, when the highlighted concern is understood to be logging.

The problem just described is not the fault of a programmer who is guilty of poor design [Simonyi, 2001]. There is simply no traditional programming language construct that would permit a better localization of the concern – it is, "a lack of expressibility in the technology available to the original designer to express interacting or overlapping concerns" [Robillard and Murphy, 2002]. Gregor Kiczales has commented that, "Many people, when they first see AOP, suggest that concerns...could be modularized in other ways, including the use of patterns, reflection, or 'careful coding.' But the proposed alternatives nearly always fail to localize the crosscutting concern. They tend to involve some code that remains in the base structure" [Kiczales, 2001]. These alternatives require that the code related to the concern be placed in numerous locations.

Aspect-Oriented Programming

Programming language support for separation of concerns has long been a core aid toward managing the complexity of large software projects. Support for the modularization and decomposition of certain dimensions of a system has improved comprehensibility and evolvability during software development. For example, objects support the decomposition of a system according to the dimensions of data abstraction and generalization (via inheritance), and structured programming techniques focus on a functional decomposition. Other dimensions of concern often concentrate on features that are crosscutting (e.g., persistence is a crosscutting feature) [Tarr et al., 1999]. Most modularization constructs, however, provide for the separation of concerns along only one dimension. The dominant form of decomposition forces other dimensions of the system to be scattered across other modules. When non-orthogonal concerns are spread out across multiple modules, the system becomes more difficult to develop, maintain, and understand. Moreover, reusability of such concerns is not possible due to the crosspollination of one concern into many modules; there is no localized container to capture the concern. As implied in the first section of this chapter, reflection and metaprogramming were an early attempt at resolving crosscutting. These techniques were somewhat low-level, but provided a lot of expressive power. With MOPs, for instance, there is a blurred distinction between language user and language designer. Therefore, a more practical use of the techniques by less experienced programmers would require modularization constructs that offered more disciplined control over this power. As these techniques evolve, a new breed of programming languages is emerging to assist in the modularization of crosscutting concerns.

Aspect-Oriented Programming (AOP) provides a strategy for dealing with emergent entities that crosscut modularity [Kiczales et al., 1997]. AOP recognizes that crosscuts are inherent in most systems and are generally not random. The goal of AOP is to provide new language constructs that allow a better separation of concerns for these aspects. An *aspect*, therefore, is a piece of code that describes a recurring property of a program that crosscuts the software application (i.e., aspects capture crosscutting concerns). AOP supports the programmer in cleanly separating components and aspects from each other by providing mechanisms that make it possible to abstract and compose them to produce an overall system.

Gregor Kiczales and his colleagues at Xerox PARC developed the seminal ideas behind AOP in the mid-1990s. In *MIT Technology Review*, AOP was featured as one of the top 10 "Emerging Technologies That Will Change the World" [Tristram, 2001] and has been

the subject of a special issue of *Communications of the ACM* [Elrad et al., 2001]. Notably, object-oriented guru Grady Booch labeled AOP as, "something deeper, something that's truly beyond objects...a disruptive technology on the horizon" [Booch, 2001].

Aspects – A Complement to Traditional Paradigms

In the structured paradigm, modular block structures were used to provide scope for separating the boundaries of concerns. The "go-to" statements that often resulted in tangled and scattered concerns were replaced with procedure calls [Dijkstra, 1968]. This improved the control flow of a program and enhanced its modularization. The Object-Oriented (OO) paradigm represents the generation that followed the structured paradigm. In OO, the key modularization technique focused on hierarchical structuring through classes and inheritance. Another key feature of OO, a polymorphism permits variation of behavior within a class hierarchy.

Each new generation of modularity technology builds upon the previous generation. AOP should be evaluated within the context of being another technology for supporting separation of concerns. The ideas of AOP should be viewed as a counterpart to procedures, packages, objects, and methods to the extent that they all support different ways of modularizing certain kinds of concerns. In this sense, AOP can be regarded as a complement to both the structured and OO paradigm, or any other paradigm for software construction (e.g., logic programming [De Volder and D'Hondt, 1999]). In AOP, the focus is on capturing, in a modular way, the crosscutting concerns of a system. The crosscuts will still exist, but the problems of scattered and tangled code are removed by encapsulating the crosscut in a single module. To quote a personal communication with Gregor Kiczales, "OO made inheritance explicit in language. AO makes crosscutting explicit in language. OO makes its bet on hierarchical structures, but AOP makes its bet on crosscutting structures."

AOP has been defined in terms of its ability to provide quantification and obliviousness. *Quantification* is the notion that a programmer can write single, separated statements that introduce effects across numerous locations in the source code. Thus, quantification would provide the capability for saying the following: "In programs P, whenever condition C arises, perform action A" [Filman, 2001]. This can be stated more formally as: C [A], where the crosscutting nature is captured in the universal quantifier and the action to be performed within the concern is the parameterized action. The property of *obliviousness* holds when the quantified locations do not require modification in order to incorporate the effects of the quantification. As stated by the authors of this definition, "AOP can be understood as the desire to make quantified statements about the behavior of programs, and to have these quantifications hold over programs written by oblivious programmers" [Filman and Friedman, 2000].

The idea of quantification does suggest a special property of aspect languages, but quantification also exists within pure metaprogramming techniques. Even though metaprogramming is one way to capture crosscutting concerns, and AOP has its roots in metaprogramming, it should be understood that there are some important differences. Perhaps a better characterization of aspect languages, in order to avoid confusion, would be those languages that provide constructs for quantification, yet do not refer to metalevel concepts.

In a first exposure to AOP, many compare it to macro expansion. However, this comparison is far from accurate. Although there are similarities with respect to code being inserted or expanded, the AOP model is much more powerful. A limitation to the strength of macros is the fact that the transformations that are performed are textually local [Kiczales et al., 1992]. For instance, to use a macro, a programmer must visit numerous locations in the source code and insert the name of the macro. If a change needs to be made, or the macro

needs to be removed from a specific context, then the programmer must visit all of these points in the code. Macros do not exhibit quantification. Aspects, on the other hand, operate under the property of reverse inheritance (also known as inversion of control2). The behavior of an aspect is specified outside of the context where it is applied. Aspects, and their quantification, are described in one location – a programmer does not have to visit and insert code in any other place. This makes the addition and removal of aspects effortless.

It should be noted that the same distinction that has been made between AOP and macros could also be made in comparing AOP and *mixins* [Bracha and Cook, 1990]. A mixin is a class that is not intended to be instantiated. It provides some desired behavior (e.g., persistence) that is imported into other classes via inheritance. Mixin-based inheritance does not provide quantification and obliviousness. If a programmer wants to include mixin behavior in a class, the mixin must be explicitly imported within the purview of the class's predecessors. Mixin based inheritance is also missing the reverse inheritance property that can be provided through the kind of quantification available in aspect languages.

In comparing aspects to classes, there is almost an inverse relation between the way inheritance works in OO and the way aspects work in AOP. As stated in [Viega and Voas, 2000], "With inheritance, classes choose what functionality they wish to subsume from other objects. Aspects, on the other hand, get to choose what functionality other objects subsume."

Examples of Commonly Recurring Crosscuts

There are several commonly recurring crosscutting concerns that have been identified from a wide variety of different systems. For example, the software described in Figure 5 highlighted the fact that the common concern of logging is often scattered across the code base.

The study of operating systems code is ripe for the mining and understanding of crosscutting concerns. As pointed out in [Coady et al., 2001b], many of the key elements of operating systems crosscut. As an illustration, the prefetching activity that is performed in OS code is often highly scattered and tangled. As Coady and colleagues discovered, the FreeBSD v3.3 implementation of prefetching was spread across 260 lines of code in 10 clusters in 5 core functions from two subsystems. A refactoring of the prefetching implementation using an aspect language demonstrated an increased comprehensibility of the code with respect to independent development, as well as the ability to (un)plug different modes of prefetching [Coady et al., 2001a]. Their future research focus is in the investigation of other crosscutting concerns in FreeBSD; namely, scheduling, communication protocols, and the file system. It is also often the case that the implementation of specific protocols lead to tangled code, as does code that is introduced into the system to improve some performance optimization. This also can be true in implementations that provide resource sharing among a set of objects. The various policies, or protocols, contained within an operating system are typically implemented in a crosscutting manner. This is similar to the observation made in Chapter 1 concerning policy implementations that have been studied in organization theory.

Perhaps the two most commonly observed crosscutting concerns are synchronization and exception handling. Both of these are also evident in the case studies of Appendix A. A detailed analysis has been performed on the ability of AOP to remove redundant code in exception handling [Lippert and Lopes, 2000]. This study looked at the code for JWAM, a framework for interactive business applications, which is implemented in over 614 Java classes in 44,000 lines of code. It was discovered that 11% of the overall code was focused on the concern of exception handling. The core of their work involved a refactoring of the exception handling code into AspectJ. The benefits of this refactorization are obvious. In many types of exceptions, they were able to reduce the amount of redundant code by a factor

of 4. Of the top five types of exceptions in the JWAM application, over 90% of the number of catch statements was removed. For example, the number of catches of the generic Exception type went from 77 in the original code to only 7 catches in the refactored AspectJ code. Similarly, the number of catches of the SQLException type went from 46 catches in the original code to only 2 in the aspectized code. Because the JWAM application was written using *Design by Contract* [Meyer, 1997], there are many assertions that test the pre- and post-conditions for a particular method. Lippert and Lopes found that over 375 post-conditions contained an assertion of "result != null" – this redundant assertion represented 56% of all post-conditions (here, redundancy referes to the replication of a single statement at the end of multiple methods). There were also 1,510 pre-conditions that contained the assertion of "arg != null"; using AspectJ, that number was cut down to 10. That is, the 1,510 pre-conditions were separated into 10 aspects, where each aspect contained a concise specification of the methods that were to contain the assertion.

The idea of superimposition, which is related to the "diffusing computation" concept initially proposed in [Dijkstra and Scholten, 1980], has recently been compared to aspect-orientation. A superimposition has been found helpful in distributed systems for maintaining and changing the global properties related to a distributed computation (e.g., deadlock detection, or the snapshot algorithm in [Chandy and Lamport, 1985]).Typically, the implementation that manages each globally distributed property is scattered in two ways: it is scattered across the processes that perform the distributed computation, and it is scattered across the source code implementation that is charged with the task of maintaining the global property. It has been noted that, "Algorithms which were intentionally designed to superimpose additional functionality on a basic program have a long history in distributed systems research, probably starting with algorithms to detect termination of basic algorithms" [Katz and Gil, 1999]. Like aspect orientation, superimpositions impose additional functionality to a base program through quantification.

Enforcing Programmer Discipline

Aspects can be used to enforce certain properties of a system that would typically be left to programmer discipline. To understand this point, reconsider the trigger example from Figure 1. Rather than using a trigger, a database administrator could have written a stored procedure, called UpdateSalary, which provides a single point of control for updating the salary field of the employee table. The UpdateSalary stored procedure could then contain, in one location, the semantics for implementing the business rule.

This solution, however, does not provide any guarantee that others will obey the rule for using only this stored procedure. There is nothing to prevent a user or developer from updating the table through means other than the stored procedure. The reliance on programmer discipline is unfeasible in large systems, and it is quite likely that certain system properties are violated when there is no direct way to enforce the concern. Aspects can be helpful in enforcing that a particular policy, or protocol, is observed in a way that does not rely on the programmer remembering to conform to a large set of unverifiable rules.

AspectJ

Early aspect languages, like COOL and RIDL [Lopes, 1997], dealt with specific types of concerns (e.g., synchronization and distribution). The most mature language, however, is a general aspect language (called AspectJ) that is an extension to Java. It is described as being general because it is not tied to capturing a particular kind of concern; instead, it provides general constructs that allow a programmer to capture a wide variety of different kinds of

concerns. The language definition has undergone many changes since the first description in [Kiczales et al., 1997] to the most recent implementation, as documented in [Kiczales et al., 2001a] and [Kiczales et al., 2001a]. This section highlights some of the key characteristics of AspectJ. AspectJ is being used in commercial development. CheckFree.com, which provides financial services for e-commerce, uses AspectJ [Miller, 2001]. An interesting anecdote is reported from this effort. A senior engineer at CheckFree stated that AspectJ allowed his team to implement a crosscutting feature in four programmer-hours. The same feature, implemented in a previous version of the application in C++, is reported to have taken two programmer-weeks [Tristram, 2001]. It has been proposed that there are three critical parts to an aspect composition language: a join point model, a way of denoting joins points, and the ability to specify behavior at those join points [Kiczales et al., 2001b].

Join Points and Pointcuts

In AOP languages like AspectJ, a *join point* denotes the location in the program that is affected by a particular crosscutting concern. This location can be either the static location of a specific line of source code, or it can represent a dynamic point during the execution of the program. There are many potential join points in a program. A *pointcut* specifies a collection of join points. The AOP literature does not provide the etymology of this term. Perhaps the intent of the terminology comes from graph theory, where the notion of a cutpoint represents a vertex in a graph whose removal would leave the graph in a disconnected state. It is a point of separation between nodes in a graph. Analogously, a pointcut is a place of potential separation for non-orthogonal concerns. A pointcut designator is declarative and permits the composition of join points using logical operators. There are many different types of pointcut designators. Several designators that will be used in a later example are:

- this(T): all join points where the currently executing object is an instance of class T
- target(T): all join points where the target object of a call is an instance of class T
- call(S): all join points (in a calling object) that are matched by a call specified by signature S
- cflow(C): this powerful designator selects all join points within the control flow of pointcut C

Advice

Whereas a join point represents a location where an aspect adds behavior, *advice* represents the behavior to add (Note: The name "advice" was chosen because it is similar to the advice feature in early Lisp machines). Advice represents a type of method that can be attached to pointcuts. The definition of an advice relates a pointcut with specific code, contained in the advice body, which takes care of the crosscutting concern. The body of the advice is normal Java code. There are three different designators for specifying the point of execution for advice: before, after, and around. The choice of these names appears to have been borrowed from CLOS [Steele, 1990]. In before advice, the advice body is executed prior to the execution of the join point's computation. The opposite is true with after advice; the advice runs after the join point computation. There are even three different kinds of after advice:

- After the successful execution of the join point (after returning);
- After an error was encountered during the execution of the join point (after throwing);
- Either of the above two cases (after).

Separation of concerns often necessitates subsequent integration. Whereas AOP provides the capability of separating numerous concerns during development, the effects of the crosscuts must be integrated back into the target code. The goal of the separation is to improve the conceptual ability of programmers during development – the end result at run-time, however, will certainly have crosscutting concerns that are transparent. As David Weiss states, in his introductory comments to one of Parnas' papers, "At run-time, one might not be able to distinguish what criteria were used to decompose the system into modules" [Hoffman and Weiss, 2001]. In AOP, a translator called a weaver is responsible for taking code specified in a traditional programming language, and additional code specified in an aspect language, and merging the two together. Because the aspect code describes numerous behaviors that crosscut a system, the concerns must eventually be integrated into the base code. This is the purpose of a weaver – it integrates aspects into the base code. In Figure 2.6, the weaving process is depicted using the previous example in Figure 2.4.

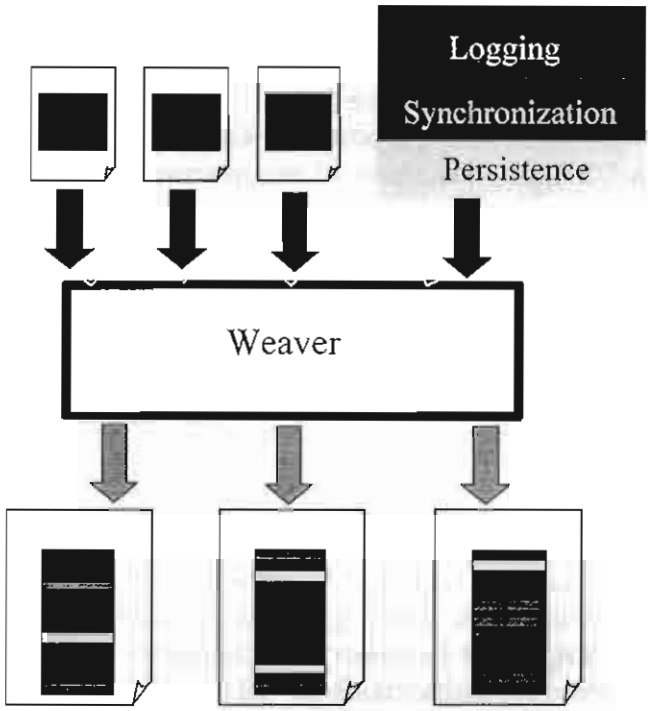


Figure 2.6: The Weaving Process

Other Work in Aspect-Oriented Software Development (AOSD)

Several researchers are working in the area of AOSD to provide new language constructs to support crosscutting concerns [Tarr et al., 1999]. Aside from AOP, other examples of specific research in this area are Subject-Oriented Programming (SOP) [Osher et al., 1996], variants of Adaptive Programming (AP) [Lieberherr et al., 2001], and Composition Filters (CF) [Bergmans and Aksit, 2001]. A hybrid approach to applying these techniques has been suggested in [Rashid, 2001]. Several of these research areas can be considered a part of generative programming, the topic of the next section.

Multi-Dimensional Separation of Concerns (MDSOC)

Another successful approach for dealing with crosscutting concerns is Subject-Oriented Programming (SOP), a research effort at IBM Research. In this approach, it is recognized that objects have different roles that they represent. These different roles can be composed into system features [Ossher et al., 1996], [Ossher and Tarr, 2001]. For example, in an Employee class, an employee object plays different roles depending on whether the Employee is being sent to the payroll subsystem (where salary and tax information are pertinent) versus the same Employee being sent to the human resources, or personnel, subsystem (where years of service and address are appropriate). The separation of these roles into isolated views is referred to as a "hyperslice" [Tarr et al., 1999]. Hyperslices assist a team of programmers in independently developing different concerns that may apply to a single class. Note that this capability was one of the Parnas' criteria described in the first chapter [Parnas, 1972].

Earlier work on subdivided procedures provided a basis for the approach adopted in SOP [Harrison and Ossher, 1990]. Subdivided procedures promote extensible programming by separating the multiple cases of procedure bodies. A procedure that dispatches from a large case statement would be an example application of subdivided procedures. In such instances, the individual cases that comprise the procedure are somewhat related to the notion of a hyperslice. An interesting comparison can be made between AOP and SOP. With AOP, the focus has always been on crosscutting concerns that are spread across multiple modules. A focus of SOP, however, has been the ability to capture several views of a single class. The separation of these views, it is argued, permits a better understanding of the implementation of each view in isolation so that the views do not become tangled. In the SOP literature, a translator called a *compositor* has numerous similarities to a weaver in AOP. A programmer creates composition rules that direct the output of the compositor [Ossher et al., 1996]. A tool called Hyper/J has been developed to support the idea of hyperslices in Java.

Adaptive Programming

The structure of objects within a class hierarchy has been found to be a type of crosscutting concern. In Adaptive Programming (AP), a key focus is the separation of behavior from structure. To aid in the modularization of this concern, visitor and traversal strategies are used [Lieberherr, 1996]. This modularization prevents the knowledge of the program's class structure from being tangled throughout the code, a desirable property that is called "structure shyness." Traversal strategies can be viewed as a specification of the class graph that does not require the hardwiring of the class structure throughout the code [Lieberherr et al., 2001]. An example of a traversal/visitor language for supporting such modularization is described in [Ovlinger and Wand, 1999]. The AP community considers their research as a special case of AOP. The motivation for AP came from the earlier work on the *Law of Demeter*, which offered a set of heuristics for improving the cohesion and coupling of object-oriented programs (the motto of this work was the anti-social message of "Talk only to your immediate friends") [Lieberherr and Holland, 1989]. In previous work at ISIS, an adaptive programming approach was used to solve a tool integration problem for a large aerospace firm [Karsai and Gray, 2000]. The domain for the integration focused on fault-analysis tools, where each tool persistently stored a model in either a database or a textual format (e.g., either comma-separated values, or a proprietary format). In that work, a model from one tool was translated into the representation of another tool. To accomplish this, semantic translators were used to traverse the graph of an internal representation of a

model. In a semantic translator, the specification of the traversal, and the actions to be performed at each traversed node, are separated.

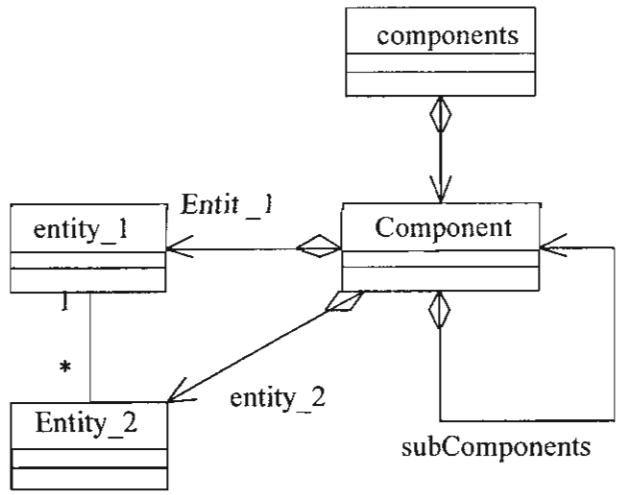


Figure 2.7: A Simple UML Tool Model Specification

The illustration in Figure 2.7 represents a simple model that is specified in the Unified Modeling Language [Booch et al., 1998]. A domain-specific language (DSL) for textually representing this diagram is presented in [Karsai and Gray, 2000]. Another DSL is shown in Figure 2.8, which demonstrates the traversal/visitor specifications that appear within a translator. During a translation, the process begins with the top model and follows along the traversal specifications. At visitor nodes, a specific action is performed that executes the

```
visitor Visitor                    visitor Visitor
{                                  {
  at Component[...]                  at Component[...]
  <<...>>                            <<...>>
   traverse[...];                     traverse[...];

  at Entity_1[...]                   at Entity_1[...]
  <<...>>;                           <<...>>;

  at Entity_2[...]                   at Entity_2[...]
  <<...>>;                           <<...>>;

  at Rel[...]                        at Rel[...]
  <<...>>                            <<...>>
   traverse[...];                     traverse[...];

}                                  }
```

Figure 2.8: Traversal/Visitor Specifications

required translation (these are elided inside of the inline code, which is denoted as <<...>>). In Figure 2.8, the first two steps in the model translation are shown by two arrows. The remaining traversal/visitor sequence would follow similarly.

Composition Filters

An earlier effort at isolating crosscutting concerns is the composition filters approach. With this technique, explicit message-level filters are added to objects and the messages that they receive [Aksit et al., 1992], [Bergmans and Aksit, 2001]. The motivation for composition filters came from the recognition that conventional object models lack the required support for separating functionality from message coordination code. As objects send messages to each other, the messages must pass through a layer of filters. Each filter has the possibility of transparently redirecting a message to other objects. Different types of filters have been found to be effective at isolating constraints and error checking [Aksit et al., 1994]. The CF approach can be very useful in executing actions before and after the interception of a method call. A related technique, proposed in [Filman et al., 2002], intercepts communication among functional components and injects behavior to support various additional capabilities (e.g., reliability, security). CORBA interceptors [Narasimhan et al., 1999] have some similarities with composition filters because they also can intercept messages. There are many exciting things on the horizon for research in aspect-oriented software development. The remainder of this section surveys some of these other research areas.

Weaver Development and Tool Support

Some of the earliest aspect languages and weavers were focused on specific concerns like synchronization and distribution. Examples of these particular aspect languages include COOL and RIDL, as defined in the dissertation of Cristina Lopes [Lopes, 1997]. More recent work, like AspectJ, has focused on generic aspect languages. Aside from Java and AspectJ, other languages are being explored with respect to AOP. The use of AspectC was cited earlier in the discussion of prefetching [Coady et al., 2001]. Although there are many difficulties in writing a C++ parser, initial efforts at providing an AspectC++ weaver (in support of real-time systems) are reported in [Gal et al., 2002], [Mahrenholz, 2002]. AspectS is an approach to general-purpose AOP in the Squeak environment [Hirschfield, 2001]. Apostle is an aspect weaver for Smalltalk [de Alwis, 2001]. A simple weaver even exists for Ruby [Bryant and Feldt, 2001]. Additionally, there has been work on making the CORBA IDL aspect-oriented [Hunleth et al., 2001], as well as efforts for bringing AOP into the realm of Microsoft .NET [Shukla et al., 2002], [Lam, 2002]. All of the weavers mentioned above are typically much more immature than the capabilities offered in AspectJ, yet they provide the major impetus for taking the ideas of AOP to other languages. In addition to weaver development, there are several other development tools that are being created to support AOP. A debugger for AspectJ, with GUI support, is available. There also has been effort to support AspectJ within several Integrated Development Environments (IDEs). Another related interesting research area is the application of AOP to compilers. As observed in [Tsay et al., 2000], "The code to do one coherent operation is spread over all node classes, making the code difficult to maintain and debug." The advantages of using AOP techniques for a weaver can be found in [de Moor et al., 1999]. In their work, the descriptions of the effects on attribute grammars are separated from the grammar productions. The benefit of this was also recognized in [VanWyk, 2000].

Debugging Aspect Code

Many aspect weavers are preprocessors that target their output code in another traditional programming language. Given the obfuscation created by the mangled names, and

the numerous indirections present in the generated code, it seems that there is a mismatch between the implementation space and the execution space. That is to say, how does a programmer write code using a particular conceptualization, and then debug the generated code that is void of that conceptualization? This question is not peculiar to AOP – the problem can be found in almost any implementation of a domain-specific language [Faith et al., 1997], [van Deursen and Knit, 1997]. To answer the question concerning the debugging of aspect code, it should be recognized that AOP is still in its early infancy. Although tool support is being developed, such as an aspect debugger, the technology is still immature. Yet, it is reasonable to expect future tools will be developed that will make the underlying execution transparent to the paradigm. In fact, the path that AspectJ is taking is not unlike the development of the earliest C++ compilers. The initial C++ compilers were merely preprocessors that generated C code. The resulting C code was void of any semblance of true object-oriented concepts – the C++ representation was merely simulated in a language that had more mature compilers. The same can be said of AspectJ and other languages concerning the incubation period needed for growth and stabilization. Perhaps a future solution to this problem will be found in an adaptation to the work in [Faith, 1997], which describes a tracking engine that interacts with a debugger and maps nodes from syntax trees.

Analysis and Design with Aspects

A study of the history of software development paradigms reveals that a new paradigm often has its genesis in programming languages and then moves out to design and analysis, or even other research areas (see [Rashid and Pulvermueller, 2000] for a description of aspects applied to databases). This same pattern also can be observed with respect to aspect-orientation. Most of the existing work on advanced separation of concerns has been heavily concentrated on issues at the coding phase of the software lifecycle. There have been, however, efforts that have focused on applying advanced separation of concerns in earlier phases of the software lifecycle. One of the first examples of this type of work can be found in [Clarke et al., 1999], where the principles of SOP were applied at the design level. Similarly, [Herrero et al., 2000] have investigated the benefits of aspects at the design level. Extensions to the UML have been proposed in order to support composition patterns as a facility for handling crosscutting requirements [Clarke and Walker, 2001], [Clarke, 2002]. A set of generic design principles for aspect-oriented software development is the focus of [Chavez and de Lucena, 2001]. An analysis of design patterns, and the aspect oriented techniques that can improve their specification and implementation, are the subject of [Nordberg, 2001]. There has been an increased interest in the need for formal verification of systems designed with support for crosscutting concerns. The most mature effort in this area can be found in [Nelson et al., 2001], where two formal languages are presented that assist in the verification of concerns focused on concurrent processes.

Aspect Mining

There is an overwhelming amount of legacy code that has been written in languages that do not support the clean separation of crosscutting concerns. To convert legacy code into languages that support AOSD, it is necessary to refactor the original program. A correct refactoring into a cleaner separation of concerns requires the examination of the original code with an eye toward aspect mining (i.e., the identification and isolation of aspects). An aspect mining tool offers assistance in this process. The Aspect Browser tool, presented in [Griswold et al., 2001], is such an example. The tool has been applied to a case study that

contained 500,000 lines of source code in FORTRAN and C. Another tool for aspect mining is described in [Hannemann and Kiczales, 2001].

AOP Validation Research

Case studies that transform legacy applications into AspectJ, like [Lippert and Lopes, 2000] and [Kersten and Murphy, 1999], provide practitioners with heuristics for adopting AOP. Both a case study and an experimental method were used in [Walker et al., 1999] to assess AOP. In an experiment that studied the ease of debugging, three synchronization errors were introduced into a Java program. A separate program that duplicated the errors was also written in AspectJ. Several teams of programmers were given the task of tracking down the errors in each of the implementations. The results of this experiment show that AspectJ provided a clear benefit to increasing localized reasoning, but no benefit when the solution required non-localized reasoning. Here, localized reasoning refers to whether or not a programmer needs to leave the context of the module (in this study, the file) that contains the error. Overall, the program teams that used AspectJ isolated and fixed the errors quicker than those who used pure Java. There are case studies that have compared the various different mechanisms for supporting advanced separation of concerns [Murphy et al., 2001]. Obviously, as AOP matures, additional studies will be needed to determine the benefits of these new approaches.

Aspect Reuse

As a large collection of different types of aspects is assembled, the idea of aspect reuse will become an interesting research topic. AOP presents new issues for reuse researchers [Grundy, 2000]. In order to be successful at aspect reuse, developers will need to begin writing their aspects in a more generic style than is currently prevalent. To see why this is so, consider the code fragments that are provided. The pointcuts of these aspects are concretized and bound specifically to the methods called DisplayError and Handle. This assumption is too strong. It may often be the case that others will want to reuse this aspect, but their code does not conform to these concrete names. To remedy this problem, a style of pointcut designation is needed such that the pointcuts of the reusable aspects are abstract. In this case, those who would wish to use and extend an abstract aspect must concretize it. In fact, AspectJ permits such designations, but its use is very infrequent in the current aspect code that is being developed. Some of the issues in support of aspect reuse and composition have been initially explored in the work on aspectual components [Lieberherr et al., 1999].

Another research issue occurs in the reuse of orthogonal aspects that apply to the same join point. This issue is important because the ordering of the generated code may be essential. For example, given the two previous aspects of locking and logging, it is often the case that, when applied to the same join point, the mutex code should appear before the logging instructions. AspectJ provides the dominates construct to allow the specification of priority between two different aspects. It is unclear, however, whether this construct alone is able to allay all of the possible problems in composing several aspects within the same join point.

Generative Programming

The first FORTRAN compiler took 18 programmer-years to complete [Backus et al., 1957]. One could argue that the time that it would take today to write an equivalent compiler would be on the order of programmer-months, not programmer-years. Of course, much of the

decreased development time would be related to the experience that has been collected on the topic of compiler construction. Most would agree, however, that the principal reason for the decreased development time would be that we have moved beyond the manual handcrafting of "one-of-a-kind" solutions to an approach that resembles an automated assembly line. To be specific, in the case of implementing a simplistic version of a FORTRAN compiler, a programmer today would use parser generators, specialized components, and perhaps even object-oriented frameworks. In implementing a compiler using modern techniques, the reduction in development time is the result of a paradigm shift toward the engineering of families of systems, as proposed in [Parnas, 1976]. The idea of a family of systems is best categorized as a domain-specific product-line architecture, where a set of different products can be created from adaptations that are made from a set of varying features [Clements and Northrop, 2001]. An excellent example of this idea is found in [Delisle and Garlan, 1990], which describes development at Tektronix on a family of oscilloscopes. An additional contributing factor to the relative ease in constructing a modern-day FORTRAN compiler is in the recognition that many of the arduous implementation details of software construction can be handed off to a generator. This paradigm shift has led toward a research area that has been dubbed Generative Programming (GP). Generative programming is accomplished by transforming higher-level representations of programs into a lower-level equivalent representation. This section surveys several of the promising research areas that are being associated with the GP movement. More detailed coverage of GP can be found in [Czarnecki and Eisenecker, 2000].

Domain-Specific Languages

A Domain-Specific Language (DSL) is a, "programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain" [van Deursen et al., 2000]. DSLs assist in the creation of programs that are more concise than an equivalent program written in a traditional programming language. An upward shift in abstraction often leads to a boost in productivity. It has been observed that a few lines of code written in a DSL can generate a hundred lines of code in a traditional programming language [Herndon and Berzins, 1988]. A key advantage is that a DSL is perspicuous to the domain expert using the language. A DSL is typically more concise because much of the intentionality of the domain is built into the generator. To use a connotation borrowed from Polya, the intent of a DSL is "pregnant with meaning" [Polya, 1957]. A DSL can assist in isolating programmers from lower-level details, such as making the decisions about specific data structures to be used in an implementation. Instead, a programmer uses idioms that are closer to the abstractions found in the problem domain. This has several advantages:

- The tedious and mundane parts of writing a program are automated in the translation from the DSL to a traditional programming language.
- Repetitive code sequences are generated automatically instead of the error-prone manual cut-and-paste method. The generation of error-prone code also has advantages during the maintenance phase of a project's lifecycle. Programs written in a DSL are usually easier to understand and modify because the intention of the program is closer to the domain.
- Solutions can be constructed quickly because the programmer can more easily focus on the key abstractions.

The size and scope of a DSL is much smaller than that of a traditional programming language. In fact, DSLs are often called "little languages" [Bentley, 1986], [van Deursen and

Knit, 1997], [Aycock, 1998]. Another common characteristic is the declarative nature of these languages. In some cases, a DSL can be viewed as a type of specification language in addition to a general purpose programming language. A DSL can be declarative because the domain provides a particular underlying interpretation. The notations and abstractions of the domain are built into the generator that synthesizes a program written in a DSL. A DSL translator can be implemented using the standard approaches for constructing a compiler or interpreter [Aho et al., 1986]. However, the majority of the literature implements DSLs with a preprocessor. Although this approach can be simpler than writing a complete compiler, it has several disadvantages. The main disadvantage is that the generated code is converted to a base programming language. This means that type checking and other compile-time tests are done outside of the domain. It also means that feedback from run-time errors are couched in terms of the base language, not the domain. A solution to this problem (previously cited in the section on "Debugging Aspect Code") is suggested in [Faith, 1997]. There are other disadvantages in using a DSL that often arise later in the development cycle. As observed in [van Deursen and Knit, 1997], the use of a DSL introduces new maintenance issues. For instance, the generators that process the programs in a DSL may often need maintenance.

Example Domains

There are numerous domains where DSLs have been applied. Some of the example domains are telecommunications [Bonachea et al., 1999], operating systems [Puet al., 1997], typesetting and drawing [Bentley, 1986], web services [Fernández et al., 1999], caching policies [Barnes and Pandey, 1999], [Gulwani et al., 2001], and databases [Horowitz et al., 1985]. The concept of a domain-specific metalangauge has also been put forth as a technique for assisting in the domain of language translators [Van Wyk, 2000]. An extensive annotated bibliography of research in this area can be found in [van Deursen et al., 2000]. Domain-specific modeling has been successfully applied in several different domains, including automotive manufacturing [Long et al., 1998], digital signal processing [Sztipanovits et al., 1998], and electrical utilities [Moore et al., 2000].

Compilers for DSLs have often been called *application generators* [Horowitz et al., 1985], [Cleaveland, 1988], [Smaragdakis and Batory, 2000]. A *generator* is a tool – a type of translator or compiler – that takes as input a domain-specific language and produces as output source code that can be compiled as a traditional programming language. The internal architecture of a generator is very similar to a compiler. A generator requires: a front-end to parse a source language into an intermediate representation, a translation engine to perform transformations and optimizations, and a back-end to produce the target code. In [Hunt and Thomas, 2000], a distinction is made between passive code generators and active code generators. In a *passive code generator*, the generator is executed just once to produce a result. After the output of a passive generator is obtained, the result becomes freestanding. The origin of the file is forgotten. An example of this type of generator would be a design wizard, like that described in [Batory et al., 2000]. With a wizard, a user enters various configuration data as a response to interacting with a dialog window. Based upon this configuration information, the wizard can then generate code that would have been tedious to create by hand. The code produced from an *active code generator*, though, frequently hanges such that it is advantageous to invoke the generator on variations of the input. There is some evidence that generators improve productivity and reliability. A comparative experiment for a Command, Control, Communication, and Information (C^3I) system is described in [Kieburtz et al., 1996]. This experiment compared the use of generators with a previously developed Ada template-based approach for implementing message translation and validation. The results of this experiment show that the teams that used the generator approach were three

times more productive than those who performed the same task using templates. The generator approach also realized improvements in reliability, with under half as many test run failures.

GenVoca

GenVoca permits hierarchical construction of software through the assembly of interchangeable/reusable components [Batory and Geraci, 1997]. The GenVoca model is based upon stacked layers of abstraction that can be composed. A *realm* is a library of plug-compatible components. It can be thought of as a catalog of problem solutions that are represented as pluggable components that can be used to build applications in the catalog domain. Each realm exposes a common interface that all components in that realm must satisfy. This provides the ability to have many alternative implementations for the same interface. The layered decomposition of implementations offers component composition that is similar to the stacking of layers in a hierarchical system. Each realm in the hierarchy is denoted by a GenVoca grammar. This grammar describes all of the legal compositions that may occur within the realm. The composition of components in GenVoca is performed by writing parameterized type expressions. These expressions are checked against the grammar to preserve validity.

A comparison between GenVoca and AOP is made in [Cardone, 1999]. Both aspect languages and GenVoca type equations guide the transformation of programs. The AOP weaver and the GenVoca generator are the preprocessors that implement such transformations. GenVoca has the capability of validating the correctness of component compositions. This is an issue that has not received much focus within the AOP research community. As mentioned in an earlier section, control over the order in which a weaver applies multiple aspects on the same join point is very limited. GenVoca, though, provides control over the ordering of component composition.

Intentional Programming

Intentional programming (IP) provides a software development environment that is not tied to a specific programming language. The power of IP is the ability to create new abstractions for languages. It allows the tailorability of a specific language to a new domain. As Charles Simonyi states, "Under IP, domain experts write models/specs/programs in domain terms" [Simonyi, 2001]. The IP system provides the functionality for defining the manner in which these new abstractions interact with the environment's text editor, as well as syntactic and semantic constructs for translating these extensions to the abstractions already supported in the IP system [Simonyi, 1996]. Thus, IP allows a programmer to write ordinary programs and domain transformations. The nodes of an Abstract Syntax Tree (AST) typically represent the semantic constructs of a language (e.g., a *while*-loop or *if*-statement). In IP, these nodes are called *intentions*. Many intentions are common across a wide variety of programming languages. The IP environment provides the capability to modify the semantics of an intention for a particular language, as well as introduce new intentions peculiar to that language. New intentions introduce their own syntax in addition to prescribing the effects of interactions with the programmer through an editor. The IP concept of an *enzyme* represents a transformation that is performed on an AST. An enzyme assists in the creation of new intentions that are built on top of existing intentions.

Parser Generators, Language Extenders, and Analysis Tools

Parser generators, like the Purdue Compiler Construction Tool (PCCTS) and YACC (Yet Another Compiler-Compiler), are programs that help in the creation of other programs that perform transformations on source code [Parr, 1993]. In the area of parser generators, an example of an extensible framework for building compilers in Python is described in [Aycock, 1998]. A framework that creates ASTs and associated tree-walker classes, based on the Visitor pattern [Gamma et al., 1995], is described in [Gagnon, 1998]. Other compiler frameworks, like Zephyr [Wang et al., 1997] and SUIF [SUIF2, 2000], provide an extensible framework to support collaborative experimental research. A primary goal of these efforts is to provide an infrastructure to benchmark different techniques that are used in compilers.

The Jakarta Tool Suite (JTS) contains the basic tools to support the addition of new programming features to the Java language [Batory et al., 1998]. It assists in the construction of new preprocessors for DSLs that are transformed into a host language. The supported host language in JTS is called Jak. Jak is described as a superset of Java that supports metaprogramming. It seems likely that JTS could be used to create a weaver for new aspect languages to support Java. The JTS environment builds upon the ideas of GenVoca. Each new extension to Java represents a new realm. Within the context of the Ptolemy project, a code generator for transforming Java programs is available [Tsay et al., 2000]. This generator is situated within an infrastructure that can parse Java programs and perform transformations on the AST using the Visitor pattern [Gamma et al., 1995].

Frameworks

A *framework* can be defined as a skeleton of an application that can be extended to produce a customized program [Fayad et al., 1999]. This type of framework is usually defined as a collection of classes that together help support a domain-specific architecture. A framework architecture must define the objects that are to participate in the framework as well as the interaction patterns among all objects. In this architecture, there is a distinction between those who create the framework and core objects (the framework developer) and the programmer who extends the framework by plugging in their own application objects (the application programmer). Frameworks typically cost more to develop than a single application, although their cost can be amortized over each instantiation [Johnson, 1997].

Adaptability in frameworks is provided by factoring out component objects that implement the core functionality in the application domain from those objects that vary with each instantiation of the framework. A framework instantiation is defined as the insertion of instance-specific classes into the framework architecture. The locations of variability within a framework are referred to as the *hot spots* of the framework [Lewis, 1995]. The instance-specific classes must conform to a predefined interface in order to properly interact with the core objects. The specification of the hot spots is needed for users of the framework because frameworks exhibit the property of inversion of control. In typical software development, the components that are written contain the locus of control in the application and selectively pass control onto other library components or lower-level calls to an Application Program Interface (API). In a framework, however, the locus of control resides in the framework, rather than the application objects. The flow of control traverses through the objects of the framework until a hot spot is reached, at which time the application object is dispatched.

Event-based infrastructures also demonstrate the principle of inversion control [Gianpaolo et al., 1998]. In an event-based approach, there is a distinction in the architecture between suppliers, consumers, and the event dispatcher. Suppliers submit events to a mediating dispatcher that forwards events to all consumer objects that have subscribed to the

event (suppliers may also be consumers of other events). The asynchronous nature of the consumers suggests a type of control inversion that provides a high degree of dynamic reconfigurability within distributed object computing. A popular example of this architecture is present in the CORBA event service [Harrison et al., 1997].

Frameworks have been developed in practically every domain that supports variability among a family of products [Fayad et al., 1999], [Fayad, 2000]. One particular interesting research area combines the topic of a previous section (AOP) with a framework for a concurrent object system [Constantinides et al., 2000].

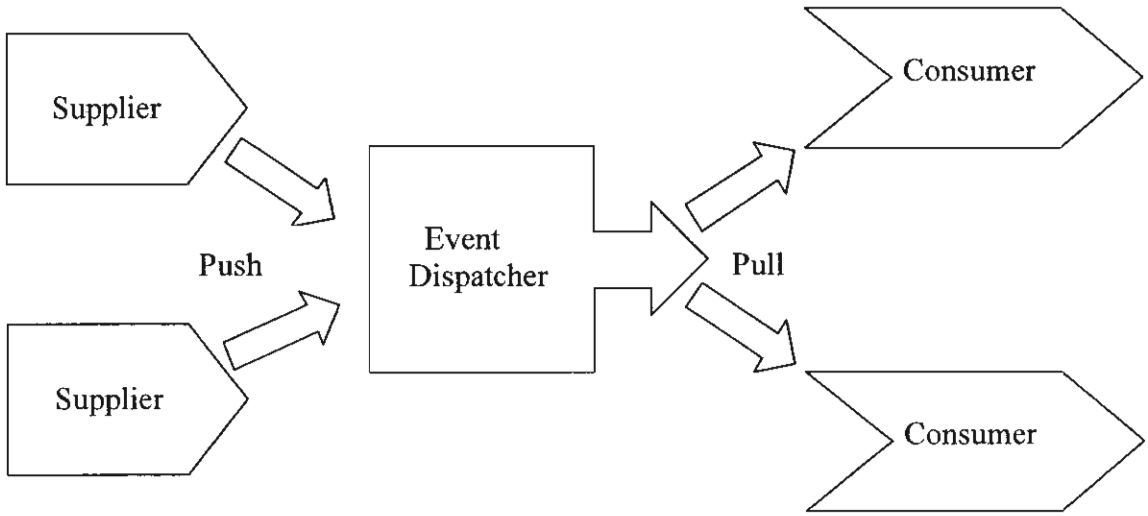


Figure 2.9: Architecture for Event-based Dispatching

Summary

This chapter provided a synopsis of the techniques that are useful in the development of software that must adapt to changing requirements. The first half of the chapter presented an overview of the literature on reflection, metaprogramming, and AOSD. The research in these areas has produced new ideas and methods for improving adaptability, and extensibility for separating crosscutting concerns. This separation provides an advantage for realizing the three objectives presented by Parnas (see "Criteria for Decomposition" in the Chapter 1). The second half of the chapter surveyed research that can be classified under the general area of Generative Programming. A generative approach captures the intent of the problem space at a higher level of abstraction. Generators map the higher abstractions to the lower-level details in the solution space. In the next two chapters, these techniques (e.g., reflection and metamodeling, advanced separation of concerns, and generative programming) will be extended to support aspect-oriented domain-specific modeling.

CHAPTER III

THE FRAMEWORK

In Aspect-Oriented Programming we decompose a problem into a number of functional components as well as a number of aspects and then we compose these components and aspects to obtain system implementations. The goal is to achieve an improved separation of concerns in both design, and implementation. Our work concentrates on the aspectual decomposition of concurrent object-oriented systems. Following the component hierarchy within the object-oriented programming paradigm we categorized aspects as intra-method, intra-object and intra-package according to their hierarchical level of cross-cutting. We achieve composition of concerns; through the use of an object we call the moderator that coordinates the interaction of components and aspects while preserving the semantics of the overall system. Since aspects can crosscut components at every level, we view the moderator is a recurring pattern from intra-method to intra-package. Our design framework provides an adaptable model and a component hierarchy using a design pattern. The moderator pattern is an architecture that allows for an open language where new aspects (specifications) can be added and their semantics can be delivered to the compiler through the moderator. In essence the moderator is a program that extends the language itself. Our goal is to achieve separation of concerns and retain this separation without having to produce an intermingled source code.

Regarding how aspects are defined and merged to provide the overall system, we believe that neither the use of aspect languages nor a weaver tool provides a necessity in order to achieve separation of concerns. We shift the weavers responsibility to a class, which we call the moderator class that would coordinate aspects and components together (figure 1). The moderator class should be extensible in order to make the overall system adaptable to addition of new aspects. We also believe that the use of a moderator class provides the flexibility, adaptability, and extensibility to the programmer to retain the definition of aspects by current programming languages. It also provides the basis for a design framework that would make use of patterns. The importance of design patterns within the AO technology was addressed in [Lorenz 98]. The moderator class defines the semantic interaction between the components and the aspects. Further, the semantics of the model define the order of activation of the aspects. We view a concurrent (shared) object as being decomposed into a set of abstractions that form a cluster of cooperating objects: a functional behavior, synchronization, and scheduling. The behavior of a concurrent object can be reused, or extended. There are other issues that might also be involved, such as security and fault tolerance. We focus on the relationships between these abstractions within the cluster. We propose an aspect-oriented design pattern that we call the aspect moderator pattern. This pattern makes use of a class, which acts as a proxy to the functional component, and would moderate the functional behavior together with different aspects of concern, by handling their interdependencies. We stress the fact that the activation order of the aspects is the most important part in order to verify the semantics of the system. Synchronization has to be verified before scheduling. A possible reverse in the order of activation may violate the semantics. If security is introduced to a shared object, we first need to verify the identity of the caller and therefore we first have to handle security before synchronization.

Architecture of the moderator pattern

A sequential object is comprised by functionality control and shared data. Access to this shared data is controlled by synchronization and scheduling abstractions.

Synchronization controls enable or disable method invocations for selection. The synchronization abstraction is composed of guards and post-actions. During the precondition phase, guards will validate the synchronization conditions. In the post-condition phase, post-actions will update the synchronization variables. The scheduling abstraction allows the specification of scheduling restrictions and terminate actions. At the pre-condition phase, scheduling restrictions use scheduling counters to form the scheduling condition for each method. At the post-condition phase, terminate actions update the scheduling counters. The moderator class is derived from the functionality class. During the pre-condition phase, the synchronization constraints of the invoked method are evaluated. If the current synchronization condition evaluates to TRUE, a RESUME value is returned to the caller, and the scheduling constraints are evaluated; otherwise a BLOCKED value is returned. The evaluation of the scheduling restrictions will also return RESUME or BLOCKED. After executing the precondition phase, the moderator will activate the method in the sequential object. During post-condition, synchronization variables and scheduling counters are updated upon method completion. This section addresses four issues: 1) non-orthogonality of aspects, 2) the provision of an adaptable model, 3) the provision of a design and implementation hierarchy and 4) composition of aspects.

Non-orthogonal aspects

The moderator can handle the issue of non-orthogonal aspects by expressing the semantics of the dependencies between two non-orthogonal aspects. For example, during the pre-condition phase of the security aspect, the moderator can include variables from any non-orthogonal aspect to security.

Extensibility and Adaptability

System software undergoes two types of evolution: functional evolution, when the problem domain changes, and adaptation, when the characteristics of the solution change. The latter is also called non-functional evolution, and it is often related to the technological changes in the applications environment. The object-oriented approach was originally developed to simplify software evolution. Unfortunately, objects are only concerned with functional evolution; they have serious problems coping with the majority of non-functional concerns, which are usually scattered in many classes, in obscure ways. Experience shows that extensibility is not a directly addressed by object-orientation: using objects does not guarantee that the software will be easily modifiable. Objects are not, therefore, the composition units we are seeking for an extensible architecture. Currently, new paradigms have emerged to deal with the intrinsic problems of objects. In particular, we have aspect-oriented programming (AOP) and component models. In aspect oriented programming, an application is built as the integration of aspects which are different solutions to different concerns. Each concern represents Aspects can be replaced, or extended. One of the advantages of this approach is that if a new aspect of concern would have to be added to the system, we do not need to modify the moderator. We can simply create a new class to inherit and re-define it, and reuse it for a new behavior. The inherited class can handle all previous aspects, together with the newly added aspect. Much like one can weave aspects on demand, such as tracing aspects [Böllert, 1998], our framework provides this option by easily adding or ignoring an aspect of a component within a cluster.

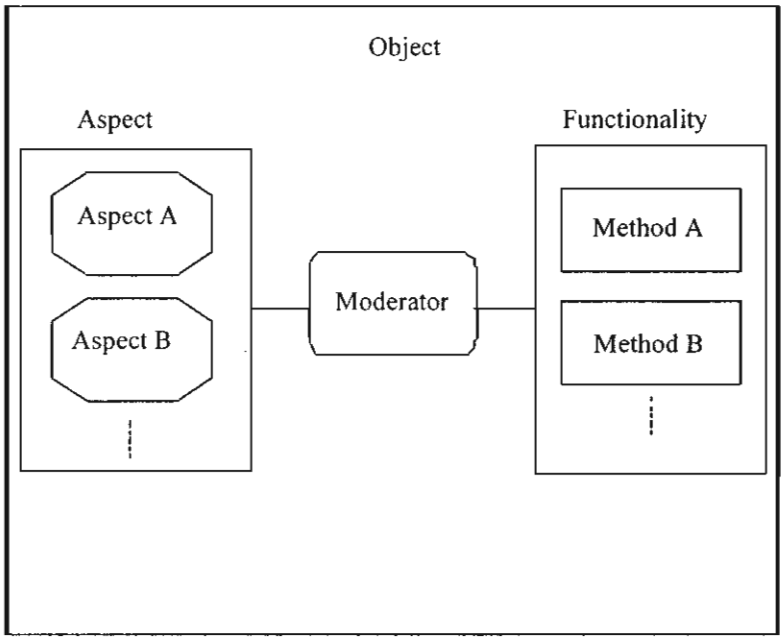


Figure 3.1. The Aspect Moderator.

Adaptability is also applied to components. This design framework addresses the complexity issue in the case where new aspects are introduced and would have to be added. The aspect-moderator pattern does not require some new syntactic structure for the representation of new aspects, but simply a new class for the new aspect. Adaptability includes cases where an existing aspect will have to be modified, or even removed from the overall system. The only composition mechanism is the functional connection, which permits to substitute different implementations of the same functionality, but is not sufficient to support unexpected evolution of the problem domain. Therefore, using components as evolution units is not completely satisfactory. We want to build applications by composition of high-level elements. Those elements are neither objects nor components, and the extension mechanism is not the simply the connection of well-defined interfaces. In itself, this goal is not new, and in the recent years interesting work has been performed to reach this objective, through different means. The following presents, in a general way, how we have reached that extensibility goal.

Design Hierarchy

Aspect Moderator seems natural to choose classes (objects) as components in the OOP paradigm. We take this argument further and propose a hierarchy of components according to the component hierarchy within the OOP paradigm. At the lowest level we have a method. Methods are combined into objects where each object belongs to a class, and several classes can belong to a package. We can apply the moderator pattern to all levels of this hierarchy since aspects can cut across every member of this component hierarchy. One or more aspects can cut across invocations within a single method. We call these aspects, intra-method (or inter-invocation). Aspects can also cut across methods within a single object. We refer to these as intra-object aspects (or inter-method). Aspects can also cut across objects within the same package. We refer to these as intra-package aspects (or inter-object). The programmer has to identify the aspects at each level and address them independently. Since

aspects can cut across components at every level, the moderator is a recurring pattern from intra-method to intra-package. Our design framework will be based on this hierarchy since we believe that it provides a better aspectual analysis and design of a system. Our approach follows a component design and implementation hierarchy.

According to our classification of aspects, the moderator at the lowest level can therefore be referred to as intra-method. At the next level the moderator is referred to as intra-object, and at the highest level it is referred to as intra-package moderator.

Composition of Aspects

At each level of the hierarchy we can maintain an aspect bank, where the moderator of a cluster may initially need to collect all the required aspects from. The aspect bank is a hierarchical 3-Dimensional composition of the system in terms of aspects and components. The moderator will initially consult the aspect bank in order to collect the required aspects.

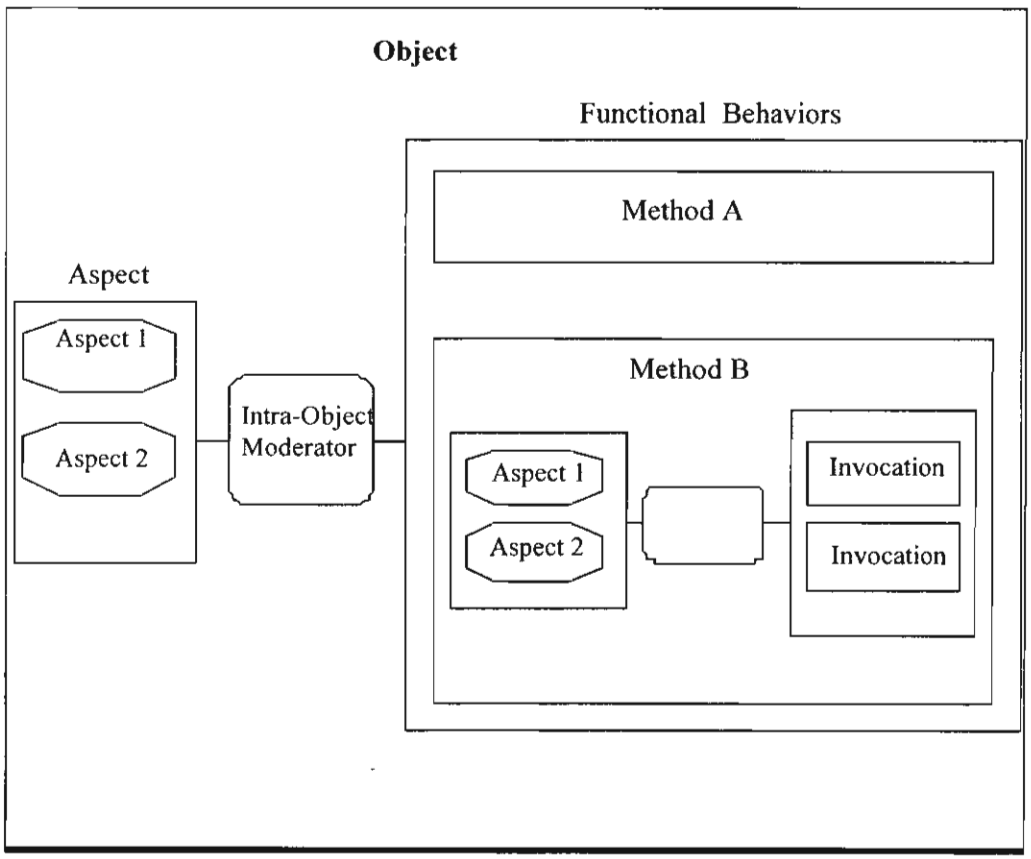


Figure 3.2. Design Hierarchy.

Example: The Conference Room Reservation System

To illustrate the rational behind the design principles of the Moderator, we present the Conference Room reservation system; an extended version of the room reservation system that presented in [Vogel and Duddy, 1998]. In this system we have components that represent rooms and employees. If a meeting organizer is interested in reserving a conference room for a meeting on a certain date and time, then the meeting organizer must check the availability of the conference room on that date and time (Figure 3.1-3.4). A new requirement states that a conference room is reserved based on the security requirements that only employee at the level of technical managers or above may reserve conference rooms. To codify this requirement, we only need to add the security aspect to the aspect bank and extend the moderator to evaluate the security aspects without the need to modify the functionality of the participating components (Figure 3.5). It is the moderator that evaluates the security code during the pre-activation phase. Therefore the moderator must be extended in order to register this new aspect for evaluation.

```
public interface  ModeratorIF {

 synchronized public int preactivation(int MethodID, Object object);
 synchronized public int postactivation(int MethodID, Object object);
 public int RegisterAspect(int MethodID, int AspectKind, AspectObject aspectObject);

}
```

Figure 3.3. The moderator interface.

```
public class AspectBank
AspectFactoryIF {
// Each method has its own aspect objects.
// The moderator evaluates the cross-cutting aspects for the involved components.
public AspectObject create (MethodID  id, AspectKind  aspect, Object Component) {
 if (id == RESERVEROOM) {
 if (aspect == SYNC)
   return new ReserveRoomSync (Component);

 }

return(0);
 }
}
```

Figure 3.4. Implementation of the aspect bank.

```
public class ConferenceRoomReservation {
// Constructor
ConferenceRoomReservation(Moderator  moderator,  AspectBank aspectBank) {
// register all aspects for each method with the moderator
moderator.RegisterAspect (SYNC, ReserveRoom,
aspectBank.create(ReserveRoom,  SYNC, this));
}

public void  ReserveRoom(int RoomId, int Date, int StartTime, int TimeWindow,
object MeetingOrganizer) {

//  PREACTIVATION PHASE : call  preactivation
// Evaluate the aspects for this method
if( moderator.preactivation(ReserveRoom, this)) == ABORT)
return ABORT;

// ACTIVATION PHASE : execute the guarded code
room[RoomId].reserver(Date,startTime,TimeWindow);
MeetingOrganizer.Update(PersonalCalendar, Date, StartTime, TimeWindow);

//  POSTACTIVATION PHASE : call  postactivation
moderator.postactivation (ReserveRoom, this);
}
}
```

Figure 3.5. Implementation of the room reservation system class.

```
synchronized public State preactivation(int MethodID, Object object) {
int AspectIndex, ComponentIndex;

// evaluate each aspect for each of the participants.
for(AspectIndex=0; AspectIndex <NoOfAspects; AspectIndex ++)
for(ComponentIndex =0;  ComponentIndex < NoOfComponents; ComponentIndex++) {
if (EvaluateAspect(AspectIndex, MethodID, ComponentIndex) == ABORT)
return ABORT;
}

// All aspects evaluated to true for all participating components.
return(RESUME);
}
```

Figure 3.6. Implementation of pre-activation.

Relation between moderator and open implementation

The moderator pattern is an architecture that allows for an open language where new aspects (specifications) can be added and their semantics can be delivered to the compiler through the moderator. In essence the moderator is a program that extends the language itself.

```
public class AspectBank2 extends AspectBank {

public AspectObject create(MethodID  id, AspectKind  aspect, Object Component){
      if (id == RESERVEROOM) {
        if (aspect == SECURITY)
        return new ReserveRoomSecurity(Component);
      }
      // the aspect may be defined in the base class
      return (super. create(id, aspect, Component));
  }
}
```

Figure 3.7. Extensibility aspect bank.

Comparison with other work

This section compares our proposal for a design framework using the moderator pattern, and current approaches that rely on the use of a weaver. Both the weaver and the moderator approaches provide the elegance of the original clean code during the analysis and design of the system. The differences between using a weaver and using the moderator pattern are summarized by the following table:

Weaver	Moderator
Combines two kinds of code (aspect and component code) into one intermingled source code. Output of the weaver is the equivalent of traditional approach.	Coordinates two kinds of code, retaining the separation of concerns (aspect and component code). Avoid having to produce an intermingled source code.
One weaver combines all aspects and components together. There is no design hierarchy.	Moderator is a recurring design pattern. It provides an overall system hierarchy, addressing intra-method, intra-object, and intra-package aspects in a systematic way.
Adding new aspect(s) will require either new aspect language(s) or new construct(s) within current aspect languages	Adding new aspect(s) is done by inheritance, and by adding new pre-condition and post-condition.
Must gather contact points of emerging entities.	A design pattern hooks aspects and components: the moderator class defines the semantic interaction between components and aspects
Two phases of compilation (weaving, compiling).	One compilation phase.

Figure 3.8. Comparison of the Framework.

Summary

In AOP, the weaver combines components (functional behavior) and aspects into one unit, which is the overall behavior of the system. In our design framework the overall behavior is made up of 1) the functional behavior, 2) the aspects, and 3) a moderator class that coordinates the interaction between aspects and components while observing the overall semantics. The Moderator approach partitions systems into a collection of cooperating classes. The collaboration among the participants may have few aspects. Addressing these aspects that cut-across the participating objects may produce tightly coupled classes which may reduce reusability. The moderator approach attempts to separate these aspects from the functional components in order to promote code reusability and make it easier to validate the design and correctness of these systems. This framework can address non-orthogonal aspects, and provide for an adaptable model with ease of modification. It further provides a component hierarchy, where the moderator is a recurring pattern. This design principle manages to achieve separation of concerns. There is no difference in the way we separate the concerns. We still have to think about them from the early stages of the software life cycle.

CHAPTER IV

REVISITED FRAMEWORK

System aspectual properties are, for instances, mutual exclusion, scheduling, synchronization, fault tolerance, logging, tracing, security, load balancing, performance measurement, testing, verifications and etc. They are all expressed in such a way that tends to crosscut groups of functional components or services. This tangling design and implementation code of system aspectual properties results increasing of code dependencies between functional components and aspectual properties of the system. It makes their source code difficult to understand, reuse, adapt, and maintain. One current attempt to resolve this issue is the Aspect-Oriented System (AOS). AOS aims at language and architecture independence, where functional components and aspectual properties are separately decomposed in both design and implementation. These properties can be captured in the design and implementation, reused, and adapted in the application software later. Finally, functional components and system aspectual properties are combined together at run-time. We distinguish between functional components and aspects in the design of systems. System aspectual properties are defined as properties of the system that do not necessarily align with functional components or services but tend to crosscut groups of functional components, increasing either *inter-dependency* or *intra-dependency*, and thus affecting the quality of the software. Intra-dependency defines as a system aspectual property that crosscuts between many services (functionalities or methods) in the same components, as illustrated in Figure 1. Inter-dependency defines as a system aspectual property that crosscuts between many components or services, as illustrated in the below figure.

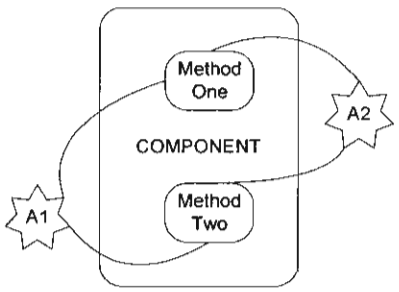


Figure 4.1. Intra-Dependency

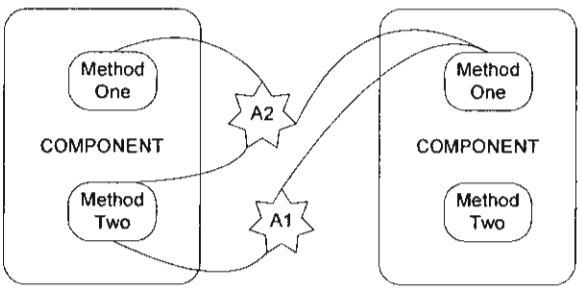


Figure 4.2. Inter-Dependency

Although not bound to OOP, Aspect-Oriented Software Development (AOSD) is a paradigm proposal that retains the advantages of OOP and aims at achieving a better separation of concerns. AOSD suggests that from the early stages of the software life cycle aspects should be addressed relatively separately from the components. As a result, aspectual decomposition manages to achieve a better design and implementation for both operating system and application. At the implementation phase, aspectual properties and functional components are combined together, forming the overall system.

In this research we have shown the system design and implementation based on system aspectual decomposition in the context of the aspectual decomposition for the design and implementation of operating systems. Our approach is an aspect-oriented framework. Compared with what has so far been able to be supported by traditional approaches, our goals are to provide a better modularity for the design and implementation of operating systems,

better flexibility, higher reusability, extensibility and adaptability, as well as to provide a technique that would be practical.

An Aspect-Oriented Framework for Operating Systems

Our observation suggests that an Aspect-Oriented Systems (AOS) that uses Aspect-Oriented Framework could support designers and programmers in cleanly separating components and system aspectual properties from each other. Our framework is based on Aspect-Oriented techniques and layered approach. We argue that system aspectual properties of the operating system should be excluded from the system components or services if there is a possibility to often change it, and it should not be treated as a single monolithic aspect.

One way of structuring system software is to decompose it into layers. Each layer is decomposed into its components. This decomposition of the system design horizontally and vertically helps to deal with the complexity and reusability of system software. The layered architectural design decomposes a system into a set of horizontal layers where each layer provides an additional level of abstraction over it's the next lower layer and provides an interface for using the abstraction it represents to a higher-level layer. Every layer is decomposed into system components and system aspectual properties. System components and system aspectual properties are separated from each other.

Changing either system components or system aspectual properties does not affect the other. The advantage of this decomposition is that system software tends to be easy to understand and maintain. Each layer can be understood and maintained individually without affecting other layers. However, it may be bad for traceability because of using lower layer components.

The framework expresses a fundamental paradigm for structuring system software, a vertical composition of each layer where system components and system aspectual properties are composed into an abstraction of the layer. The framework uses a client-server model in which the server components (Functional Components and System Aspectual Components) are composed by the Aspect Moderator and make their services available to clients. Clients access the server component services by sending requests to the Proxy component. The Proxy component intercepts a requesting message from clients and forwards the message to Aspect Moderator component. The Aspect Moderator component locates and instantiates the composition rules defined by pointcut(s) – where consist of join points between functional components and system aspectual components.

The aspect-oriented framework supports both vertical and horizontal compositions. Functional and aspectual property components in the framework can be composed vertically or horizontally. In vertical composition, the upper layer can use the lower functional or aspectual property components from the lower layer. In horizontal composition, functional and aspectual property components in the particular layer only use to be composed.
The framework is based on system aspectual decomposition of crosscutting concerns in operating system design and implementation.

The framework consists of two frameworks: The Based Layer and The Application Layer Framework. A system aspectual property is implemented in the SystemAspect class, while a component of the system is implemented as a Component class. Alike AspectJ, our framework uses *PointCut, Precondition,* and *Advice.* The framework uses *PointCut, Precondition,* and *Advice.* The AspectModerator class, where the point cut is defined, combines both system aspectual properties and components together at runtime. Pointcuts are defined collections of join points, where system aspectual properties will be altered and executed in the program flow. Every aspectual property can identify and implement preconditions. A precondition is defined a set of conditions or requirements that must hold in

order that an aspect may be executed. Advice is a defined collection of methods for each aspectual property that should be executed at join points. Advice can be either *before* or *after* advice. Before advice can be implemented as *blocking* or *non-blocking*. Before advice is executed when the join point is reached, before the component is executed, if the precondition holds. After advice is executed after the component at the join point is executed. Every aspectual property will define advice methods. Figure 4.3 and 4.4 illustrated the execution model of a pointcut in the framework based on inter-dependency and intra-dependency.

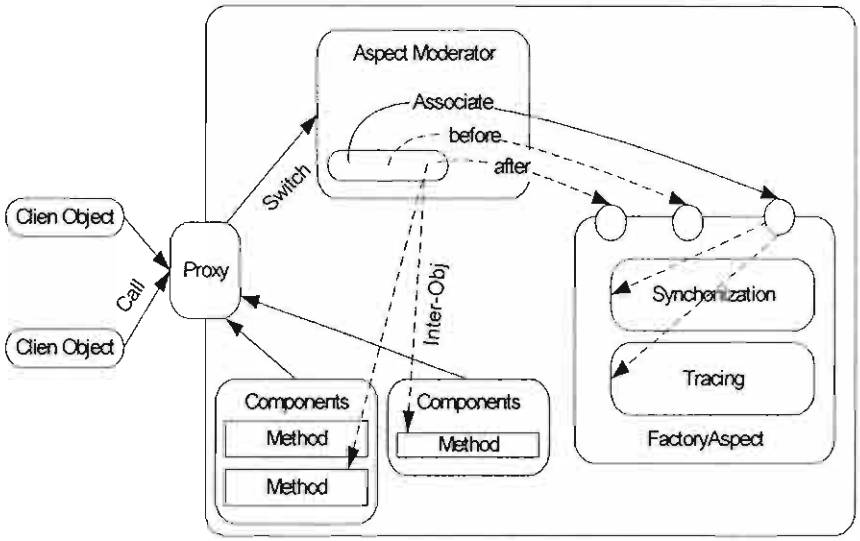


Figure 4.3. PointCut Defines Inter-dependency

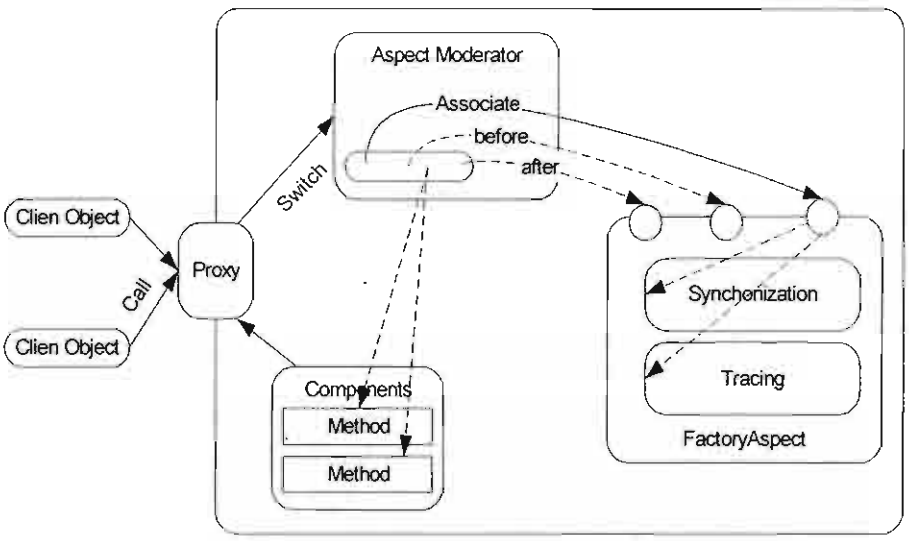


Figure 4.4. PointCut Defines Intra-dependency

Our proposed framework (CAL) is based on system aspectual decomposition of crosscutting concerns in operating system design and implementation. ACL framework consists of two frameworks: Based Layer and Application Layer Framework. In this paper, we show how producers/consumers problem can be implement in the based layer framework. A system aspectual property is implemented in SystemAspect class, while a component of the system is implemented as Component class. AspectModerator object, where the point cut is defined, combines both system aspectual properties and components together at run-time. A Pointcut is defined collections of join points, where system aspectual properties will be altered and executed in the program flow. Every aspectual property could identify and implement precondition. Precondition is defined a set of conditions or requirements that must be hold in order to be executed an aspect. Advice is defined collections of methods for each aspectual property that should be executed at join points. Advice could be either before or after. Before advice could be implemented as blocking or non-blocking. Before advice executes when join point is reached, before the component executed, and if the precondition is hold. After advice executes after the component at the join point executes.

Implementing Aspect-Oriented Framework

The framework consists of four components comprising the architecture of the framework.

- Each functional object (component) provides its services (methods) stripped of any aspectual properties (for example, no synchronization is included in Buffer objects).
- A proxy object intercepts called methods and transfers the calls to the AspectModerator.
- An AspectModerator object consists of the rules and strategies needed to bind aspects at runtime. Aspects are selected from the AspectBank. The AspectModerator orders the execution of aspects. The order of execution can be static or dynamic. Then, each precondition will be checked whether it is satisfied or not.
- An AspectBank object consists of aspect objects that implement different policies of a variety of aspects.

This section presents the design and development of aspect-oriented framework. The model is presented to demonstrate horizontal composition of the framework. The system service must be implemented as a Component class. The system aspectual property (SystemAspect class) must be derived from the SystemAbstractAspect interface to implement the required behavior of a system aspectual property. A SystemAspectFactory consists of many system aspectual properties such as synchronization, tracing, logging, and reliability. The SystemAspectFactory, derived from the SystemAbstract.

AspectFactory interface, is known as an aspect bank. During runtime, each SystemAspectFactory will be associated with one SystemAspect. The AspectModerator class must be derived from the AspectModerator interface to implement the required behavior.

The following points are important about the aspect-oriented framework:

- A base layer framework is an implementation of an underlying system.
- An application layer framework is an implementation of application software over the system software represented by a base layer framework.
- A client object requests a service through a ProxyObject object of a framework.
- A functional component is implemented as a Component class without any aspectual property.
- A SystemAspectFactory object consists of various SystemAspect objects. A SystemAspect object is controlled by a SystemAspectFactory object.

- Each system aspectual property must be implemented as a SystemAspect object.
- Each crosscutting between Component object and an SystemAspect object must be defined in AspectModerator object as joinpoints in a Pointcut method.
- A client requests a service by sending a message to a ProxyObject object. The ProxyObject object changes the request to a specific pointcut method, and forwards it to the AspectModerator object.

The Proxy class is responsible for intercepting and forwarding the message sent from client object to request a service. The Proxy class must implement the behavior of intercepting a service request. A client object of an aspect-oriented framework must request a service by calling the call() method. A call() method consists of at least two parameters: object name provided a service and a service requested to serve. The first parameter is of type string, and the second is type of string as well. The ProxyObject class will forward a request to the AspectModerator object by calling a PointCut() method. A PointCut() method must have the same number parameters and the same parameter type as the call() method The SystemAspectFactor class must be derived from the SystemAspectFactoryAbstract interface to implement the required behavior.

The AspectModerator class is responsible for composing the functional components and the system aspectual property into a service request. The AspectModerator class acts like a coordinator between functional components and system aspectual properties, when and where system aspectual properties will be composed into a functional component. The composition of system aspectual properties and functional components must be guided and defined as PointCut() method. Each PointCut() method must have at least two parameters: component name and service name (methods of the component) that will be composed. The first parameter is of type string, and the second is type of string as well.

The SystemAspectFactor class must be derived from the SystemAspectFactoryAbstract interface to implement the required behavior. The SystemAspectFactory class provides a dynamic binding of variety system aspectual properties. It focuses on the interface of the system aspectual property. Each system aspectual property must be derived from the SystemAspectAbstract interface to implement the required behavior. Implementation of a system aspectual property is implemented in the SystemAspect class. Each system aspectual property can define before(), after(), and precondition() methods depending on its needs. Figure 10 demonstrates the system aspectual property (SystemAspect class) declaration determined from the base class SystemAspectAbstract.

The AspectModerator class operates composition between system aspectual properties and functional components using a composition rule defined by join points of a pointcut. The AspectModerator class performs composition rules by sending AspectFactory messages. Messages sending causes polymorphism. The implementation of AspectFactory uses bridge patterns. A message finds the correct member object of the AspectFactory, and invokes that object. With polymorphism calls, AspectModerator requires less information about each SystemAspect, so the AspectModerator only needs to have the right SystemAspect interface.

The abstract aspectual class defines a SystemAbstractAspect interface that controls the implementation of an aspectual property class. This class is implemented using the concrete classes of aspectual properties, which implement the virtual functions before() and after(). The AspectModerator creates instances of an aspectual property, which requires composing a requested service. If an aspectual property crosscuts more than one method in the same component, it must have a parameter ServiceName identifying what it should be done for each method. If an aspectual property crosscuts more than one component, it must have two parameters: ServiceName and ComponentName identifies what it should be done for each method of each component.

Summary

In this research, we stressed the importance of the better separation of concerns within the context of an Aspect-Oriented Framework. We discussed how this technique provide an alternative to operating system design and implementation, and show how our approach can be achieved separation of crosscutting concerns in the design and implementation of operating systems. Our work concentrates on the decomposition of system aspectual properties crosscutting functional components in the system and our goal is to achieve a better design and implementation of operating systems while supporting separation the crosscutting concerns in every layer. Our design framework provides an adaptable model that allows for open languages and architectures where new aspects and components can be easily manageable and added without invasive changes or modifications. In application, system aspectual properties could be reused and redefined from the system layer preventing the re-engineering of all aspects and components. The framework approach is promising, as it seems to be able to address a large number of system and application aspects and components. The advantage of decomposing of functional components and aspects makes the design and implementation of operating systems better modularity as well as is to promote comprehension, reusability, adaptability, manageability, and extensibility of both components and aspects in the system.

CHAPTER V

CONCLUSION

The object-oriented approach was originally developed to simplify software volution. Unfortunately, objects are only concerned with functional evolution; they have nous problems coping with the majority of non-functional concerns, which are usually ttered in many classes, in obscure ways. Experience shows that extensibility is not a ectly addressed by object-orientation: using objects does not guarantee that the software ll be easily modifiable. Objects are not, therefore, the composition units we are seeking for extensible architecture. Currently, new paradigms have emerged to deal with the intrinsic oblems of objects. In particular, we have aspect-oriented programming (AOP) and mponent models. Each concern represents a problem facet. The basic idea is to define, parately, on the one hand the application logic and on the other hand the different concerns, d to weave them all later on.

In component-oriented programming, the basic structure of the architecture is a graph here nodes are (black box) components described by means of their functional capacities, d the links represent the provide/require relationship. The only missing decision is the ncrete implementation of each one of those boxes. Thanks to this decoupling, local olution inside one component is well supported, but global evolution (a change of the aph structure) or adaptability is hardly handled. The only composition mechanism is the nctional connection, which permits to substitute different implementations of the same nctionality, but is not sufficient to support unexpected evolution of the problem domain. herefore, using components as evolution units is not completely satisfactory.

AOP approach has several advantages but it poorly supports evolution because the eaving is performed directly on the language structures that can evolve. The application and aspects are too closely related. The underlying problem is that in the AOP architecture here are no composition elements, but only a mechanism for code weaving. For this reason, e do not consider that AOP proposes an extensible architecture.

We have identified important issues in the design of adaptable and extensible erating systems, the complexity of system comprehension, development, reusability, tensibility and adaptability. Functional components and system aspectual properties, such mutual exclusion, synchronization, fault tolerance, and tracing aspects, are not well parated using current operating system design. This prevents the designer and developer m understanding, modifying, extending, adapting, and reusing the components of the stem.

To solve these issues, we developed an aspect-oriented framework for the design of tensible and adaptable operating systems. The framework is designed based on the concept Aspect-Oriented Software Development. It allows designers and programmers to separate nctional components and system aspectual properties from each other in every component.

We have shown implementation of classical problems using an aspect-oriented mework. An aspect-oriented design framework simplifies system design by expressing it at higher level of abstraction.

trength of This Research

As with the architecture of a building, the excellence of a software structure or design not easy to measure. Many researchers and developers use the attribute comprehension,

Comprehensibility

Comprehension is a measure of how easy it is for a designer and a programmer to understand the design and implementation of the system. System aspectual properties crosscut basic functional components of the system. With consistency of the design and implementation, system aspectual properties can be captured in both the system design and implementation. We believe that an aspect-oriented framework supports the designers and programmers in cleanly separating functional components and system aspectual components from each other, by providing a mechanism that makes it possible to abstract and compose both functional components and system aspectual components to produce the overall system. Both functional components and system aspectual components can be easily understood.

We believe that the framework provides a better separation of concerns in the design of operating systems. The framework promotes better modularity and quality in the design of the system. The design of operating systems should not be seen as a two-dimensional model with a single monolithic aspectual property. In this research we stress the importance of the complete separation of concerns as proposed by Aspect-Oriented Software Development and we discuss how this methodology can provide an alternative approach to operating system design. Our approach simplifies system design by expressing it at a higher level of abstraction using a three-dimensional model. It further supports the designers and programmers in cleanly separating functional components and system aspectual components from each other in different layers.

Adaptability

Adaptability is a measure of how flexible, modifiable, and easily extensible it is for a designer and a programmer to adapt the existing system. With better separation of concerns, adaptability or refinement of either functional components or system aspectual components of the system can further be achieved easily.

Adding or changing functional components does not affect system aspectual components at all. On the other hand, adding or changing system aspectual components does not affect functional components either. Only Pointcuts, defined in the AspectModerator component, are modified; thus, system aspectual properties that crosscut functional components will not be affected.

Applicability

Applicability refers to the utility of the framework for its intended use. The framework is primarily designed to be an alternative for the design of the adaptable operating systems with better separation of concerns, reuse, and adaptability. Indeed, the design that is good for one software package or application may be poor for others, and conversely. The framework solves complexity of both adaptability of functional components and system aspectual components.

Scalability and Expansibility

Expansibility is a measure of how easy it is for a designer or a programmer to increase or scale the capability of functional components and system aspectual components. The framework supports horizontal and vertical scalability and expansibility. From experimental

studies, the framework allows a designer and a programmer to use a wide range of functional components and system aspectual components in the Aspect Moderator component.

Horizontal extension was identified as a way to extend the process functionalities by additional and specialized features. We realized that handling extensibility at a pretty high level of abstraction was possible, and was bringing in many very interesting possibilities.

Vertical extension, in contrasts, is in charge of bridging the gap between the abstract level and the real tools and services used to implement the high level functions. The experience gained showed us that these two very different mechanisms are complementary and both needed to build complex systems. The approach was used to realize experiences addressing other issues related with process technology. These experiments proved to be surprisingly successful.

With horizontal scalability and expansibility, designers or programmers can achieve scalability and expansibility of functional components or system aspectual components in a variety of ways: They can only increase or drop functional components, they can only increase or drop system aspectual components, or they can increase or drop both components. With vertical scalability and expansibility, designers and programmers can achieve scalability and expansibility of layers.

Reusability

Reusability refers to reuse of prefabricated and standard components for increasing productivity, standard, cost, and time. It also provides easier maintenance and consistency when the implementation changes, but the interface is still the same. With reusability, the developers can reduce complexity.

Because software design is complex and costly, reusing the design framework can be applied as a blue print for the design of adaptable operating systems. Using the framework, as a development foundation means that we do not have to start from scratch each time we design and develop the system. Frameworks provide a way to reuse the design and implementation. Reusing an abstraction is always easier than inventing one. A programmer can reuse reusable and extensible functional and system aspectual components from the lower layer that may be utilized by other programmers. The abstraction of an aspect component in the lower level provides transparency. The upper aspectual components or functional components can use the lower aspect components or functional components without knowing the internal details of how the lower aspect components or functional components are implemented. Information hiding promotes either functional or aspectual component modifiability, and simplifies the perception of the upper level. The upper level components or applications can use the abstraction of aspectual and functional components in the lower level without knowing the internal details of how the lower level aspect is constructed. If a lower aspect is changed (to improve performance or to add new features, for example), provided the aspect interface (intermediate level) remains constant, the upper level aspect need not change. This approach may result in better modifiability of the system.

REFERENCES

[Aho et al., 1986] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.

[Aksit et al., 1992] Mehmet Aksit, Lodewijk Bergmans, and S. Vural, "An Object-Oriented Language-Database Integration Model: The Composition Filters Approach," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 615, Springer-Verlag, Utrecht, The Netherlands, June/July 1992, pp. 372-395.

[Aksit et al., 1994] Mehmet Aksit, Jan Bosch, William van der Sterren, and Lodewijk Bergmans, "Real-Time Specification Inheritance Anomalies and Real-Time Filters," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 821, Springer-Verlag, Bologna, Italy, July 1994, pp. 386-407.

[Anderson and Hickey, 1999] Kenneth R. Anderson and Timothy J. Hickey, "Reflecting Java into Scheme," *Proceedings of Reflection '99: Metalevel Architectures and Reflection,* LNCS 1616, Springer-Verlag, Saint-Malo, France, July 1999, pp. 154-174.

[AOSD, 2002] http://aosd.net

[Aycock, 1998] John Aycock, "Compiling Little Languages in Python," *Proceedings of the 7th International Python Conference*, Houston, Texas, November 1998, pp. 69-77.184

[Backus et al., 1957] J. W. Backus, R. J. Beeber, S. Best, R. Goldberg, L. M. Haibt, H. L. Herrick, R. A. Nelson, D. Sayre, P. B. Sheridan, H. Stern, I. Ziller, R. A. Hughes, and R. Nutt, "The FORTRAN Automatic Coding System," *WesternJoint Computer Conference*, 1957, pp. 188-198.

[Barnes and Pandey, 1999] J. Fritz Barnes and Raju Pandey, "CacheL: Language Support for Customizable Caching Policies," *Proceedings of the Fourth International Web Caching*
Workshop, San Diego, California, March 1999.

[Batory and Geraci, 1997] Don Batory and Bart J. Geraci, "Composition Validation and Subjectivity in GenVoca Generators," *IEEE Transactions on Software Engineering*, February 1997, pp. 67-82.

[Batory et al., 1998] Don Batory, Bernie Lofaso, and Yannis Smaragdakis, "JTS: Tools for Implementing Domain-Specific Languages," *Fifth International Conference on Software Reuse*, Victoria, Canada, June 1998, pp. 143-153.

[Batory et al., 2000] Don Batory, Gang Chen, Eric Robertson, and Tao Wang, "Design Wizards and Visual Programming Environments for GenVoca Generators," *IEEE Transactions on Software Engineering*, May 2000, pp. 441-452.

[Bergmans and Aksit, 2001] Lodewijk Bergmans and Mehmet Aksit, "Composing Crosscutting Concerns using Composition Filters," *Communications of the ACM*, October 2001, pp. 51-57.

[Bentley, 1986] Jon Bentley, "Programming Pearls: Little Languages," *Communications of the ACM*, August 1986, pp. 711-721.

[Bloch, 2001] Joshua Bloch, *Effective Java Programming Language Guide*, Addison-Wesley, 2001.

[Bonachea et al., 1999] Dan Bonachea, Kathleen Fisher, Anne Rogers, and Frederick Smith, "Hancock: A Language for Processing Very Large-Scale Data," *USENIX Conference on Domain-Specific Languages*, Austin, Texas, October 1999, pp. 163-176.

[Booch et al., 1998] Grady Booch, Ivar Jacobson, James Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.

[Booch, 2001] Grady Booch, "Through the Looking Glass," *Software Development Magazine*, July 2001, pp. 49-51.

[Bobrow et al., 1993] Daniel G. Bobrow, Richard Gabriel, and Jon L. White, "CLOS in Context: The Shape of the Design Space," A. Paepcke, editor, *Object-Oriented Programming: The CLOS Perspective*, 1993, pp. 29-61.

[Bracha and Cook, 1990] Gilad Bracha and William Cook, "Mixin-based Inheritance," *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Ottawa,
Canada, October 1990, pp. 308-311.

[Brooks, 1995] Fred Brooks, *The Mythical Man-Month*, Addison-Wesley, 1995.

[Bryant and Feldt, 2001] Avi Bryant and Robert Feldt, "AspectR – Simple Aspect-Oriented Programming in Ruby," http://aspectr.sourceforge.net

[Cardone, 1999] Richard Cardone, "On the Relationship of Aspect-Oriented Programming and GenVoca," *Workshop on Institutionalizing Software Reuse*, Austin, Texas, January 1999.

[Chandy and Lamport, 1985] K. Mani Chandy and Leslie Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Transactions on Computer Systems*, February 1985, pp. 63-75.

[Chavez and de Lucena, 2001] Christina von Flach G. Chavez and Carlos J. P. de Lucena, "Design-level Support for Aspect-Oriented Software Development," *OOPSLA Workshop on Advanced Separation of Concerns*, Minneapolis, Minnesota, October 2001.

[Chiba and Masuda, 1993] Shigeru Chiba and Takashi Masuda, "Designing an Extensible Distributed Language with a Metalevel Architecture," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 707, Springer-Verlag, Kaiserslautern, Germany, July 1993, pp. 482-501.

[Chrysler and Escobar, 2000] John Chrysler and Thomas Escobar, *2000 Masonry Codes and Specifications*, CRC Press, 2000.

[Clarke, 2002] Siobhán Clarke, "Extending Standard UML with Model Composition Semantics," *Science of Computer Programming*, May 2002.

[Clarke and Walker, 2001] Siobhán Clarke and Robert J. Walker, "Composition Patterns: An Approach to Designing Reusable Aspects," *International Conference on Software Engineering (ICSE)*, Toronto, Ontario, Canada, May 2001, pp. 5-14.

[Clavel, 2000] Manuel Clavel, *Reflection in Rewriting Logic: Metalogical Foundations and Metaprogramming Applications*, CSLI Publications, 2000.

[Cleaveland, 1988] J. Craig Cleaveland, "Building Application Generators," *IEEE Software*, July 1988, pp. 25-33.

[Clements and Northrop, 2001] Paul Clements and Linda Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[Coady et al., 2001a] Yvonne Coady, Gregor Kiczales, Mike Feeley, and Greg Smolyn, "Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code," *Proceedings of the Joint European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9)*, Vienna, Austria, September 2001, pp. 78-88.

[Coady et al., 2001b] Yvonne Coady, Gregor Kiczales, Mike Feeley, Norm Hutchinson, and Joon Suan Ong, "Structuring Operating System Aspects," *Communications of the ACM*, October 2001, pp. 79-82.

[Constantinides et al., 2000] Constantinos Constantinides, Atef Bader, Tzilla Elrad, P. Netinant, and Mohamed Fayad, "Designing an Aspect-Oriented Framework in an Object-Oriented Environment," *ACM Computing Surveys*, March 2000.

[Czarnecki and Eisenecker, 2000] Krzysztof Czarnecki and Ulrich Eiseneker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[de Alwis, 2001] Brian de Alwis, "Apostle: Aspect Programming in Smalltalk," http://www.cs.ubc.ca/~bsd/apostle/

[De Volder and D'Hondt, 1999] Kris De Volder and Theo D'Hondt, "Aspect-Oriented Logic Meta Programming," *Proceedings of Reflection '99: Metalevel Architectures and Reflection,* LNCS 1616, Springer-Verlag, Saint-Malo, France, July 1999, pp. 250-272.

[Daft et al., 1987] Richard Daft, Kristen Skivington, and Mark Sharfman, *Cases and Applications: Organizational Theory*, West Wadsworth, 1987.

[Date, 1999] C.J. Date, *An Introduction to Database Systems*, Addison-Wesley, 1999.

[Delisle and Garlan, 1990] Norman Delisle and David Garlan, "Applying Formal Specification to Industrial Problems: A Specification of an Oscilloscope," *IEEE Software*, September 1990, pp.29-36.

[Dessler, 1986] Gary Dessler, *Organization Theory: Integrating Structure and Behavior*, Prentice-Hall, 1986.

[Dijkstra, 1968] Edsger Dijkstra, "Go To Statement Considered Harmful," letter to the editor, *Communications of the ACM*, March 1968, pp. 147-148.

[Dijkstra, 1976] Edsger Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976.

[Dijkstra and Scholten, 1980] Edsger Dijkstra and C.S. Scholten, "Termination Detection for Diffusing Computations," *Information Processing Letters,* August 1980, pp. 1-4.

[Elrad et al., 2001] Tzilla Elrad, Mehmet Aksit, Gregor Kiczales, Karl Lieberherr, and Harold Ossher, "Discussing Aspects of AOP," *Communications of the ACM*, October 2001, pp. 33-38.

[Faith, 1997] Rickard Edward Faith, *Debugging Programs After Structure-Changing Transformations*, Ph.D. Dissertation, Department of Computer Science, The University of North Carolina at Chapel Hill, 1997.

[Faith et al., 1997] Rickard E. Faith, Lars S. Nyland, and Jan F. Prins, "Khepera: A System for Rapid Implementation of Domain-Specific Languages," *USENIX Conference on Domain-Specific Languages*, Santa Barbara, California, October 1997, pp. 243-255.

[Fayad et al., 1999] Mohamed Fayad, Douglas Schmidt, and Ralph Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, John Wiley and Sons, 1999.

[Fayad, 2000] Mohamed Fayad, "Introduction to the Computing Surveys' Electronic Symposium on Object-Oriented Application Frameworks," *ACM Computing Surveys*, March 2000.

[Fernández et al., 1999] Mary Fernández, Dan Suciu, and Igor Tatarinov, "Declarative Specification of Data-intensive Web Sites," *USENIX Conference on Domain-Specific Languages*, Austin, Texas, October 1999, pp. 135-148.

[Filman and Friedman, 2000] Robert Filman and Dan Friedman, "Aspect-Oriented Programming is Quantification and Obliviousness," *OOPSLA Workshop on Advanced Separation of Concerns*, Minneapolis, Minnesota, October 2000.

[Filman, 2001] Robert Filman, "What is Aspect-Oriented Programming, Revisited," *ICSE Workshop on Advanced Separation of Concerns*, Toronto, Ontario, Canada, May 2001.

[Filman et al., 2002] Robert Fillman, Stuart Barrett, Diana Lee, and Ted Linden, "Inserting Ilities by Controlling Communications," *Communications of the ACM*, January 2002, pp. 116-122.

[Forman and Danforth, 1999] Ira R. Forman and Scott H. Danforth, *Putting Metaclasses to Work*, Addison-Wesley, 1999.

[Gabriel, 1995] Richard P. Gabriel, "The Column Without a Name: Software Development as Science, Art and Engineering," *C++ Report*, July/August 1995.

[Gabriel and Goldman, 2000] Richard P. Gabriel and Ron Goldman, "Mob Software: The Erotic Life of Code," *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Keynote Address, Minneapolis, Minnesota, October 19, 2000.

[Gagnon, 1998] Etienne Gagnon, "SableCC: An Object-Oriented Compiler Framework," Master's Thesis, School of Computer Science, McGill University, Montreal, March 1998.

[Gal et al., 2002] Andreas Gal, Wolfgang Schröder-Preikschat, and Olaf Spinczyk, "On Aspect-Orientation in Distributed Real-Time Dependable Systems," *IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, San Diego, Califorinia, January 2002.

[Gamma et al., 1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Gianpaolo et al., 1998] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta, "Exploiting an Event-Based Infrastructure to Develop Complex Distributed Systems," *International Conference on Software Engineering (ICSE)*, Kyoto, Japan, April 1998, pp. 261-270.

[Gray et al., 2000] Jeff Gray, Ted Bapty, and Sandeep Neema, "Aspectifying Constraints in Model-Integrated Computing," *OOPSLA Workshop on Advanced Separation of Concerns*, Minneapolis, Minnesota, October 2000.

[Gray, 2001a] Jeff Gray, "Using Software Component Generators to Construct a Metaweaver Framework," *International Conference on Software Engineering (ICSE)*, Toronto, Ontario, Canada, May 2001, pp. 789-790.

[Gray, 2001b] Jeff Gray, "A Framework for Creating Aspect Weavers," *Doctoral Symposium: OOPSLA '01 Companion to Proceedings*, Tampa, Florida, October 2001.

[Gray et al., 2001a] Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck, "Handling Crosscutting Constraints in Domain-Specific Modeling," *Communications of the ACM*, October 2001, pp. 87-93.

[Gray et al., 2001b] Jeff Gray, Ted Bapty, and Sandeep Neema, "An Example of Constraint Weaving in Domain-Specific Modeling," *OOPSLA Workshop on Domain-Specific Visual Languages*, Tampa, Florida, October 2001, pp. 49-56.

[Griswold et al., 2001] William G. Griswold, Jimmy J. Yuan, and Yoshikiyo Kato, "Exploiting the Map Metaphor in a Tool for Software Evolution," *International Conference on Software Engineering (ICSE)*, Toronto, Ontario, Canada, May 2001, pp. 265-274.

[Grundy, 2000] John Grundy, "Multi-Perspective Specification, Design and Implementation of Software Components Using Aspects," *International Journal of Software and Knowledge Engineering*, December 2000, pp. 713-734.

[Gudmundson and Kiczales, 2001] Stephan Gudmundson and Gregor Kiczales, "Addressing Practical Software Development Issues in AspectJ with a Pointcut Interface," *ECOOP Workshop on Advanced Separation of Concerns*, Budapest, Hungary, June 2001.

[Gulwani et al., 2001] Sumit Gulwani, Aasha Tarachandani, Deepak Gupta, Dheeraj Sanghi, Luciano Barreto, Gilles Muller, and Charles Consel, "WebCaL - A Domain-Specific Language for Web Caching," *Computer Communications*, February 2001, pp. 191-201

[Hall, 1998] Richard Hall, *Organizations: Structure, Process, and Outcomes*, Prentice-Hall, 1998.

[Hannemann and Kiczales, 2001] Jan Hannemann and Gregor Kiczales, "Overcoming the Prevalent Decomposition in Legacy Code," *ICSE Workshop on Advanced Separation of Concerns*, Toronto, Ontario, Canada, May 2001.

[Harrison and Ossher, 1990] William Harrison and Harold Ossher, "Subdivided Procedures: A Language Extension Supporting Extensible Programming," *International Conference on*
Computer Languages, New Orleans, Louisiana, March 1990, pp. 190-197.

[Harrison et al., 1997] Timothy Harrison, David Levine, and Douglas C.Schmidt, "The Design and Performance of a Real-Time CORBA Event Service," *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Atlanta, Georgia, October 1997, pp. 184-200.

[Herndon and Berzins, 1988] Robert M. Herndon and Valdis Berzins, "The Realizable Benefits of a Language Prototyping Language," *IEEE Transactions on Software Engineering*, June 1988, pp. 803-809.

[Herrero et al., 2000] Jose Herrero, Fernando Sanchez, Fabiola Lucio, and Migeul Torro, "Introducing Separation of Aspects at Design Time," *ECOOP Workshop on Aspects and Dimensions of Concerns*, Cannes, France, June 2000.

[Hirschfield, 2001] Robert Hirschfield, "AspectS – AOP with Squeak," *OOPSLA Workshop on Advanced Separation of Concerns*, Tampa, Florida, October 2001.

[Hoffman and Weiss, 2001] Daniel Hoffman and David Weiss, editors, *Software Fundamentals – Collected Papers by David L. Parnas*, Addison-Wesley, 2001.

[Hoftstadter, 1979] Douglas R. Hofstadter, *Gödel, Escher, Bach*, Random House, 1979.

[Horowitz et al., 1985] Ellis Horowitz, Alfons Kemper, and Balaji Narasimhan, "A Survey of Application Generators," *IEEE Software*, January 1985, pp. 40-54.

[Hunleth et al., 2001] Frank Hunleth, Ron Cytron, and Chris Gill, "Building Customized Middleware Using Aspect-Oriented Programming," *OOPSLA Workshop on Advanced Separation of Concerns*, Tampa, Florida, October 2001.

[Hunt and Thomas, 2000] Andrew Hunt and David Thomas, *The Pragmatic Programmer*, Addison-Wesley, 2000.

[Hunt and Thomas, 2002] Andy Hunt and Dave Thomas, "Software Archaeology," *IEEE Software*, March/April 2002, pp. 20-22.

[IEEE 1471, 2000] *IEEE Standard 1471-2000: Recommended Practice for Architectural Description of Software-Intensive Systems*, The Institute for Electrical and Electronics Engineers, Inc., October 2000.

[Johnson, 1997] Ralph E. Johnson, "Frameworks = (Components + Patterns)," *Communications of the ACM*, October 1997, pp. 39-42.

[Karr et al., 2001] David Karr, Craig Rodrigues, Joseph Loyall, Richard Schantz, Yamuna Krishnamurthy, Irfan Pyarali, and Douglas Schmidt, "Application of the QuO Quality-of-Service Framework to a Distributed Video Application," *International Symposium on Distributed Objects and Applications*, Rome, Italy, September 2001.

[Karsai and Gray, 2000] Gábor Karsai and Jeff Gray, "Component Generation Technology for Semantic Tool Integration," *IEEE Aerospace Conference*, Big Sky, Montana, March 2000.

[Katz and Gil, 1999] Shmuel Katz and Joseph Gil, "Aspects and Superimpositions," *ECOOP Workshop on Aspect-Oriented Programming*, Lisbon, Portugal, June 1999.

[Kersten and Murphy, 1999] Mik Kersten and Gail C. Murphy, "Atlas: A Case Study in Building a Web-based Learning Environment Using Aspect-Oriented Programming," *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Denver,Colorado, November 1999, pp.340-352.

[Kiczales et al., 1991] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow, *The Art of the Metaobject Protocol*, MIT Press, 1991.

[Kiczales, 1992] Gregor Kiczales, "Towards a New Model of Abstraction in the Engineering of Software," *Proceedings of the International Workshop on New Models for Software Architectures (IMSA): Reflection and Metalevel Architecture*, Tokyo, Japan, November 1992, pp. 1-11.

[Kiczales et al., 1992] Gregor Kiczales, John Lamping, Luis H. Rodriguez Jr., and Erik Ruf, "Macros that Reach Out and Touch Somewhere," Internal Technical Report, Embedded Computation Area, Xerox PARC, 1992.

[Kiczales et al., 1993] Gregor Kiczales, J. Michael Ashley, Luis Rodriguez, Amin Vahdat, and Daniel G. Bobrow, "Metaobject Protocols: Why We Want Them and What Else Can They Do?" A. Paepcke, editor, *Object-Oriented Programming: The CLOS Perspective*, 1993, pp. 101-108.

[Kiczales, 1996] Gregor Kiczales, "Beyond the Black Box: Open Implementation," *IEEE Software*, January 1996, pp. 8-11.

[Kiczales et al., 1997] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241, Springer-Verlag, Jyväskylä, Finland, June 1997, pp.220-242.

[Kiczales, 2001] Gregor Kiczales, "Aspect-Oriented Programming: The Fun Has Just Begun," *Software Design and Productivity Coordinating Group – Workshop on New Visions for Software Design and Productivity: Research and Applications*, Nashville, Tennessee, December 2001.

[Kiczales et al., 2001a] Gregor Kiczales, Eric Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William Griswold, "An Overview of AspectJ," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 2072, Springer-Verlag, Budapest, Hungary, June 2001, pp.327-353.

[Kiczales et al., 2001b] Gregor Kiczales, Eric Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William Griswold, "Getting Started with AspectJ," *Communications of the ACM*, October 2001, pp. 59-65.

[Kieburtz et al., 1996] Richard B. Kieburtz, Laura McKinney, Jeffrey M. Bell, James Hook, Alex Kotov, Jeffrey Lewis, Dino P. Oliva, Tim Sheard, Ira Smith, and Lisa Walton, "A Software Engineering Experiment in Software Component Generation," *International Conference on Software Engineering (ICSE)*, Berlin, Germany, March 1996, pp.542-552.

[Knuth, 1984] Donald Knuth, "Literate Programming," *The Computer Journal*, May 1984, pp. 97-111.

[Lam, 2002] John Lam, "Cross-Language Load-Time Aspect Weaving on Microsoft's Common Language Runtime," Demonstration, *First International Conference on Aspect-Oriented Software Development*, Enschede, The Netherlands, April 2002.

[Lee and Zachary, 1995] Arthur H. Lee and Joseph L. Zachary, "Reflections on Metaprogramming," *IEEE Transactions on Software Engineering*, November 1995, pp. 883-893.

[Lego, 2002] http://www.lego.com/eng/info/profile.asp

[Lewis, 1995] Ted Lewis, ed., *Object-Oriented Application Frameworks*, Manning Publications, 1995.

[Lieberherr and Holland, 1989] Karl Lieberherr and Ian Holland, "Assuring Good Style for Object-Oriented Programs," *IEEE Software*, September 1989, pp. 38-48.

[Lieberherr, 1996] Karl Lieberherr, *Adaptive Object-Oriented Software*, International Thomson Publishing, 1996.

[Lieberherr et al., 1999] Karl Lieberherr, David Lorenz, and Mira Mezini, "Programming with Aspectual Components," NU-CCS-99-01, College of Computer Science, Northeastern University, March 1999.

[Lieberherr et al., 2001] Karl Lieberherr, Doug Orleans, and Johan Ovlinger, "Aspect-Oriented Programming with Adaptive Methods," *Communications of the ACM*, October 2001, pp. 39-41.

[Lieberman, 1986] Henry Lieberman, "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems," *Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, Portland, Oregon, November 1986, pp. 214-223.

[Lippert and Lopes, 2000] Martin Lippert and Cristina V. Lopes, "A Study on Exception Detection and Handling Using Aspect-Oriented Programming," *International Conference on Software Engineering (ICSE)*, Limmerick, Ireland, June 2000, pp. 418-427.

[Long et al., 1998] Earl Long, Amit Misra, and Janos Sztipanovits, "Increasing Productivity at Saturn," *IEEE Computer*,August 1998, pp. 35-43.

[Lopes, 1997] Cristina Lopes, *D: A Language Framework forDistributed Programming*, Ph.D. Dissertation, College of Computer Science, Northeastern University, November 1997.

[Loyall et al., 2001] Joseph Loyall, Richard Schantz, John Zinky, Partha Pal, Richard Shapiro, Craig Rodrigues, Michael Atighetchi, David Karr, Jeanna Gossett, and Christopher Gill, "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications," *IEEE International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, Arizona, April 2001.

[Maes, 1987] Pattie Maes, "Concepts and Experiments in Computational Reflection," *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Orlando, Florida, December 1987, pp. 147-155.

[Maes, 1988] Pattie Maes, "Issues In Computational Reflection," P.Maes and D. Nardi, editors, *Metalevel Architectures and Reflection*, Elsevier Science, 1988, pp. 21-35.

[Maguire, 1994] Steve Maguire, *Debugging the Development Process*, Microsoft Press, 1994.

[Mahrenholz, 2002] Daniel Mahrenholz, Olaf Spinczyk, and Wolfgang Schröder-Preikschat, "Program Instrumentation for Debugging and Monitoring with AspectC++," *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Washington, DC, April 2002.

[Maroti et al., 2002] Miklos Maroti, Ákos Lédeczi, Arpad Bakay, Jeff Gray, and Gábor Karsai, "Type Hierarchies in Modeling and Metamodeling Languages," in preparation, 2002.

[Meyer, 1997] Bertrand Meyer, *Object-Oriented Software Construction*, Prentice-Hall, New Jersey, 1997.

[Meyer, 2000] Erica Meyer, *Cascading Style Sheets: The Definitive Guide*, O'Reilly & Associates, 2000.

[Michels, 1915] Robert Michels, *Political Parties: The Sociological Study of the Oligarchical Tendencies of Modern Democracy*, translated by Eden and Cedar Paul, Batoche Books, 1915.

[Miller, 2001] Sandra Kay Miller, "Aspect-Oriented Programming Takes Aim at Software Complexity," *IEEE Computer*, April 2001, pp. 18-21.

[Moore et al., 2000] Michael Moore, Saeed Monemi, and Jianfeng Wang, "Integrating Information Systems In Electric Utilities," *IEEE International Conference on Systems, Man, and Cybernetics*, Nashville, Tennessee, October 2000.

[Murphy et al., 1999] Gail C. Murphy, Robert J. Walker, and Elisa L.A. Baniassad, "Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-Oriented Programming," *IEEE Transactions on Software Engineering*, July/August 1999, pp. 438-455.

[Murphy et al., 2001] Gail C. Murphy, Albert Lai, Robert J. Walker, and Martin P. Robillard, "Separating Features in Source Code: An Exploratory Study," *International Conference on Software Engineering (ICSE)*, Toronto, Ontario, Canada, May 2001, pp. 275-284.

[Narasimhan et al., 1999] Priya Narasimhan, Louise Moser, and P.M. Melliar-Smith, "Using Interceptors to Enhance CORBA," *IEEE Computer*, July 1999, pp. 62-68.

[Nelson et al., 2001] Torsten Nelson, Donald Cowan, and Paulo Alencar, "Supporting Formal Verification of Crosscutting Concerns," *Reflection 2001: The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, LNCS 2192, Springer-Verlag, Kyoto, Japan, September 2001, pp. 153-169.

[Nordberg, 2001] Martin Nordberg, "Aspect-Oriented Dependency Inversion," *OOPSLA Workshop on Advanced Separation of Concerns*, Tampa, Florida, October 2001.

[Ossher et al., 1996] Harold Ossher, Matthew Kaplan, A. Katz, William Harrison, and Vincent Kruskal, "Specifying Subject-Oriented Composition," *Theory and Practice of Object Systems*, vol. 2(3), 1996, pp. 179-202.

[Ossher and Tarr, 2001] Harold Ossher and Peri Tarr, "Using Multidimensional Separation of Concerns to (Re)Shape Evolving Software," *Communications of the ACM*, October 2001, pp. 43-50.

[Ovlinger and Wand, 1999] Johan Ovlinger and Mitchell Wand, "A Language for Specifying Recursive Traversals of Object Structures," *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Denver, Colorado, November 1999, pp. 70-81.

[Parnas, 1972] David Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, December 1972, pp. 1053-1058.

[Parnas, 1976] David Parnas, "On the Design and Development of Program Families," *IEEE Transactions on SoftwareEngineering*, January 1976, pp. 1-9.

[Parnas, 1999] Nancy Eickelman, "ACM Fellow: David Lorge Parnas," *ACM Software Engineering Notes*, May 1999, pp. 10-14.

[Parr, 1993] Terrence J. Parr, *Language Translation Using PCCTS and C++*, Automata Publishing Company, 1993.

[Perrow, 1986] Charles Perrow, *Complex Organizations: A Critical Essay*, McGraw-Hill, 1986.

[Polya, 1957] George Polya, *How to Solve It*, Princeton University Press, 1957.

[Rao, 1991] Ramana Rao, "Implementational Reflection in Silica," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 512, Springer-Verlag, Geneva, Switzerland, July 1991, pp. 251-266.

[Rashid and Pulvermueller, 2000] Awais Rashid and Elke Pulvermueller, "From Object-Oriented to Aspect-Oriented Databases," *Proceedings of the 11th International Conference on Database and Expert Systems Applications*, September 2000, London, UK, pp. 125-134.

[Rashid, 2001] Awais Rashid, "A Hybrid Approach to Separation of Concerns: The Story of SADES," *Reflection 2001: The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, LNCS 2192, Springer-Verlag, Kyoto, Japan, September 2001, pp. 231-249.

[Rashid, 2002] Awais Rashid, "Weaving Aspects in a Persistent Environment," *ACM SIGPLAN Notices*, February 2002, pp. 36-44.

[Robertson and Brady, 1999] Paul Robertson and J. Michael Brady, "Adaptive Image Analysis for Aerial Surveillance," *IEEE Intelligent Systems*, May/June 1999, pp. 30-36.

[Robillard and Murphy, 2002] Martin Robillard and Gail Murphy, "Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies," *International Conference on Software Engineering (ICSE)*, Buenos Aires, Argentina, May 2002.

[Schach, 2002] Stephen R. Schach, *Object-Oriented and Classical Software Engineering*, 5th ed., McGraw-Hill, 2002.

[Schmidt et al., 2000] Doug Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, Wiley and Sons, 2000.

[Shukla et al., 2002] Dharma Shukla, Simon Fell, and Chris Sells, "Aspect-Oriented Programming Enables Better Code Encapsulation and Reuse," *MSDN Magazine*, March 2002, pp. 60-68.

[Siegel, 2000] Jon Siegel, *CORBA 3 Fundamentals and Programming*, John Wiley & Sons, 2000.

[Simon, 1996] Herbert Simon, *The Sciences of the Artificial*, The MIT Press, 1996.

[Simonyi, 1996] Charles Simonyi, "Intentional Programming: Innovation in the Legacy Age," Presented at *IFIP WG 2.1*, June 1996.

[Simonyi, 2001] Charles Simonyi, "Intentional Programming: Asymptotic Fun?" *Software Design and Productivity Coordinating Group – Workshop on New Visions for Software Design and Productivity: Research and Applications*, Nashville, Tennessee, December 2001.

[Smaragdakis and Batory, 2000] Yannis Smaragdakis and Don Batory, "Application Generators," J. Webster (ed.), *Encyclopedia of Electrical and Electronics Engineering*, John Wiley and Sons, 2000.

[Smith, 1776] Adam Smith, *An Inquiry into the Nature and Causes of the Wealth of Nations*, republished in Edwin Cannan's annotated edition, 1904, Methuen and Co.; first edition, 1776.

[Smith, 1982] Brian Smith, "Reflection and Semantics in Procedural Languages," Technical Report 272, Massachusetts Institute of Technology, Laboratory for Computer Science, 1982.

[Sobel and Friedman, 1996] Jonathan M. Sobel and Daniel P. Friedman, "An Introduction to Reflection-Oriented Programming," *Reflection '96*, San Francisco, California, April 1996.

[Sommerville and Sawyer, 1997] Ian Sommerville and Peter Sawyer, "Viewpoints: Principles, Problems and a Practical Approach to Requirements Engineering," *Annals of Software Engineering*, March 1997, pp. 101-130.

[Steele and Sussman, 1978] Guy Lewis Steele, Jr., and Gerald Jay Sussman, "The Art of the Interpreter, or the Modularity Complex (Parts Zero, One, and Two)," MIT Artificial Intelligence Memo 453, May 1978.

[Steele, 1990] Guy L. Steele, Jr., *Common Lisp: The Language*, Digital Press, 1990.

[Steele, 1998] Guy Steele, "Growing a Language," *Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA)*, Keynote Address, Vancouver, British Columbia, Canada, October 22, 1998.

[SUIF2, 2000] *The SUIF 2 Compiler System*, http://suif.stanford.edu/suif/suif2/

[Sullivan, 2001] Gregory T. Sullivan, "Aspect-Oriented Programming using Reflection and Metaobject Protocols," *Communications of the ACM*, October 2001, pp. 95-97.

[Sutton and Rouvellou, 2001] Stanley M. Sutton and Isabelle Rouvellou, "Issues in Design and Implementation of a Concern-Space Modeling Schema," *ICSE Workshop on Advanced Separation of Concerns*, Toronto, Ontario, Canada, May 2001.

[Sztipanovits and Karsai, 1997] Janos Sztipanovits and Gábor Karsai, "Model-Integrated Computing," *IEEE Computer*, April 1997, pp. 10-12.

[Sztipanovits et al., 1998] Janos Sztipanovits, Gábor Karsai, and Ted Bapty, "Self-Adaptive Software for Signal Processing," *Comunications of the ACM*, May 1998, pp. 66-73.

[Tarr et al., 1999] Peri Tarr, Harold Ossher, William Harrison, and Stanley Sutton, "N Degrees of Separation: Multi-Dimensional Separation of Concerns," *International Conference on Software Engineering (ICSE)*, Los Angeles, California, May 1999, pp. 107-119.

[Tekinerdogan, 2000] Bedir Tekinerdogan, *Synthesis-Based Software Architecture Design*, Ph.D. Dissertation, Department of Computer Science, University of Twente, 2000.

[Tidwell, 2001] Doug Tidwell, *XSLT*, O'Reilly and Associates, 2001.

[Tristram, 2001] Claire Tristram, "The Technology Review Ten: Untangling Code," *MIT Technology Review*, January 2001.

[Tsay et al., 2000] Jeff Tsay, Christopher Highlands, and Edward Lee, "A Code Generation Framework for Java Component-Based Designs," *International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, San Jose, California, November 2000.

[van Deursen and Knit, 1997] Arie van Deursen and Paul Klint, "Little Languages: Little Maintenance?" *First ACM SIGPLAN Workshop on Domain-Specific Languages,* Paris, France, January 1997, pp. 109-127.

[van Deursen et al., 2000] Arie van Deursen, Paul Klint, and Joost Visser, "Domain-Specific Languages: An AnnotatedBibliography," *ACM SIGPLAN Notices*, June 2000, pp. 26-36.

[Van Wyk, 2000] Eric Van Wyk, "Domain-Specific Meta Languages," *ACM Symposium on Applied Computing*, Como, Italy, March 2000, pp. 799-803.

[Viega and Voas, 2000] John Viega and Jeffrey Voas, "Can Aspect-Oriented Programming Lead to More Reliable Software?" *IEEE Software*, November/December 2000, pp. 19-21.

[Walker et al., 1999] Robert J. Walker, Elisa L.A. Baniassad, and Gail C Murphy, "An Initial Assessment of Aspect-Oriented Programming," *International Conference on Software Engineering (ICSE)*, Los Angeles, California, May 1999, pp. 120-130.

[Wang et al., 1997] Daniel Wang, Andrew Appel, Jeff Korn, and Chris Serra, "The Zephyr Abstract Syntax Description Language," *USENIX Conference on Domain-Specific Languages*, Santa Barbara, California, October 1997, pp. 213-228.

[Weber, 1946] Max Weber, "Bureaucracy," in Hans Gerth and C. Wright Mills, eds., *From Max Weber*, Oxford University Press, 1946.

[Wegner, 1976] Peter Wegner, "Programming Languages – The First 25 Years," *IEEE Transactions on Computers*, December 1976, pp. 1207-1225.

[Wulf and Shaw, 1973] William Wulf and Mary Shaw, "Global Variables Considered Harmful," *ACM SIGPLAN Notices*, February 1973, pp. 28-34.

APPENDIX A

FRAMEWORK TEMPLATE

```
/*
 * client.h
 */

#include "proxy.h"

class clientObject
{
public:

        ....

 protected:
        ProxyObject *proxyObj;
        ....
};

ReturnType ClientObject::MethodName()
{
  // Do something useful ...

  //Request a service
  proxyObj->Call(Component_Name,  Service_Name , Return_Data);

  return ReturnType;
}



/*
 * proxy.h
 */

#include <string.h>
#include "AspectModerator.h"


class ProxyObject {

public:
        //Constructor ProxyObject
        ProxyObject(){ };

        //Service call interceptor
        bool Call(const char *, const char * ,..... );

private:
        //Create AspectModerator Object
        AspectModerator amObject;
};

bool ProxyObject::Call (const char * Component_Name, const char * Serive_Name, .....)
{
        //Forward a request to AspectModerator object
        amObject.PointCut (Component_Name, Service_Name, ..... );
        return TRUE;
}
```

```
/*
 * Component.h
 */

class componentObject
{
public:

        ....

protected:

        ....
};

void componentObject::componentObject( )
{
        ....
}

void componentObject::operationOne(....)
{
        ....
}

void componentObject::operationTwo(....)
{
        ....
}


/*
 * SystemAbstractAspect.h
 */

class SystemAbstractAspect {
public:
        virtual void before(const char * ServiceName, ......) { };
        virtual void after(const char * ServiceName, ......) { };
private:
        virtual bool precondition(const char * ServiceName, ......) = 0;
};


/*
 * SystemAbstractFactory.h
 */

#include <string.h>

class SystemAbstractAspectFactory {
public:
        virtual void precondition() = 0;
};
```

```
/*
 *SystemAspectFactory.h
 */

#include "SystemAspectAbstractFactory.h"
#include "SystemAspect.h"
#include "AspectOne.h"

class SystemAspectFactory : public SystemAbstractAspectFactory {
public:
        SystemAspectFactory();
        void attachImpl(SystemAbstractAspect *);
        virtual void precondition(const char * ServiceName, ......);
        virtual void before(const char * ServiceName, ......);
        virtual void after(const char * ServiceName, ......);

private:
        SystemAbstractAspect *pr_;
};

SystemAspectFactory::SystemAspectFactory()
{
        pr_ = 0;
}

void SystemAspectFactory::attachImpl(SystemAbstractAspect *pr)
{
        pr_ = pr;
}

void SystemAspectFactory::precondition(const char * ServiceName, ......)
{
        pr_->precondition(ServiceName, ...);
}
void SystemAspectFactory::before(const char * ServiceName, ......)
{
        pr_->before(ServiceName, ...);
}

void SystemAspectFactory::after(const char * ServiceName, ......)
{
        pr_->after(ServiceName, ...);
}
```

```
/*
 *AspectOne.h
 */

#include "SystemAbstractAspect.h"

class nameAspect : public SystemAbstractAspect {
public:
        AspectOne();

        virtual void before(const char * ServiceName, ……);
        virtual void after(const char * ServiceName, ……);
        virtual bool precondition(const char * ServiceName, ……);

private:

        ....
};

AspectOne::AspectOne()
{
}

bool AspectOne::precondition(const char * ServiceName, ……)
{
        return ….;
}


void AspectOne::before(const char * ServiceName, ……)
{
        ....
}

void AspectOne::after(const char * ServiceName, ……)
{
        ....
}
```

```
/*
 * AspectModerator.h
 */

#include <string.h>
#include "SystemFactory.h"
#include "Sync1.h"
#include "AspectOne.h"

class AspectModerator {
public:
   //Constructor
      AspectModerator() {    //Attached aspectual properties to AspectFactory
                              sysAspectFactoryObj[0].attachImpl (&r);
                            sysAspectFactoryObj[1].attachImpl (&r1);
                        };

      bool PointCut (const char *, const char * , .....);

private:
      //Declare AspectFactory
      SystemAspectFactory sysAspectFactoryObj[2];

      //Decalre Funcational Components
      component functionalObject;

      //Decalre Aspectual Properties
      AspectOne r;
      AspectTwo r1;
};


bool AspectModerator::PointCut(const char * CoponentName, const char * ServiceName, ......)
{
   //if Funcational Component = FileBuffer
      if ( strcmp(ComponentName, "FileBuffer") == 0 ) {

      //if ServiceName = WriteFile
            if ( strcmp(ServiceName, "WriteFile") == 0 ) {

         //Compose Sychronization aspect before executing the service of a functional component
                  sysAspectFactoryObj[0].before(ServiceName, obj);

            //Execute Service of Functional Component
                  functionalObject.WriteFile(obj);

                  //Compose Sychronization aspect after executing the service of a functional
component
                  sysAspectFactoryObj[0].after(ServiceName, obj);

                  //Compose Tracing aspect after executing the service of a functional component
                  sysAspectFactoryObj[1].after(ServiceName, obj);
            }
            else

      //if ServiceName = WriteFile
            if (strcmp(sFuncName,"ReadFile") == 0) {

         //Compose Sychronization aspect before executing the service of a functional component
                  sysAspectFactoryObj[0].before(sFuncName, obj);

            //Execute Service of Functional Component
                  functionalObject.ReadFile(obj);

                  //Compose Sychronization aspect after executing the service of a functional
component
            sysAspectFactoryObj[0].after(sFuncName, obj);
```

```
            //Compose Tracing aspect after executing the service of a functional component
                    sysAspectFactoryObj[1].after(sFuncName, obj);

            }
    }
    return 0;
}
```

APPENDIX B

READERS/WRITERS PROBLEM USING OBJECT-ORIENTATION

```
/*
 * writer.h
 */

#include "buffer.h"
#define WIN32_LEAN_AND_MEAN

typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
  LPVOID lpThreadParameter
  );
typedef unsigned *PBEGINTHREADEX_THREADID;

// This ThreadObject is created by a thread
// that wants to start another thread. All
// public member functions except ThreadFunc()
// are called by that original thread. The
// virtual function ThreadMemberFunc() is
// the start of the new thread.
//

class WriterObject
{
public:
        WriterObject();
  void StartThread(BufferObject &);
  void WaitForExit();

  static DWORD WINAPI WriterFunc(LPVOID param);

protected:
  virtual DWORD WriterMemberFunc();
        DWORD getThreadId() {return m_ThreadId; };

  HANDLE  m_hThread;
        DWORD   m_ThreadId;
        BufferObject *bufferObj;
};

WriterObject::WriterObject()
{
  m_hThread = NULL;
  m_ThreadId = 0;
        BufferObject *bufferObj = new BufferObject();
}


void WriterObject::StartThread(BufferObject & obj)
{
        bufferObj = &obj;
        m_hThread = (HANDLE)_beginthreadex(NULL,
    0,
    (PBEGINTHREADEX_THREADFUNC) WriterObject::WriterFunc,
    (LPVOID)this,
    0,
    (PBEGINTHREADEX_THREADID) &m_ThreadId );
        if (m_hThread) {
    cout << "Thread WRITER Id: " << getThreadId() << " launched" << endl;
  }
}



void WriterObject::WaitForExit()
{
// WaitForSingleObject(m_hThread, INFINITE);
        DWORD threadExitCode;
```

```
        GetExitCodeThread(m_hThread, &threadExitCode);
        cout << "Thread WRITER Id: " << getThreadId()
                << " Exit Code = " << threadExitCode << endl;
    CloseHandle(m_hThread);
}

DWORD WINAPI WriterObject::WriterFunc(LPVOID param)
{
    // Use the param as the address of the object
    WriterObject* pto = (WriterObject*)param;
    // Call the member function. Since we have a
    // proper object pointer, even virtual functions
    // will be called properly.
    return pto->WriterMemberFunc();
}


//
// This above function ThreadObject::ThreadFunc()
// calls this function after the thread starts up.
//
DWORD WriterObject::WriterMemberFunc()
{
    // Do something useful ...
        bufferObj->WriteFile(m_ThreadId);
    return 0;
}
```

```
/*
 * reader.h
 */

#include "buffer.h"
#define WIN32_LEAN_AND_MEAN

typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
  LPVOID lpThreadParameter
);
typedef unsigned *PBEGINTHREADEX_THREADID;

/*
 * This ThreadObject is created by a thread
 * that wants to start another thread. All
 * public member functions except ThreadFunc()
 * are called by that original thread. The
 * virtual function ThreadMemberFunc() is
 * the start of the new thread.
 */
class ReaderObject
{
public:
  ReaderObject();
  void StartThread(BufferObject &);
  void WaitForExit();

  static DWORD WINAPI ReaderFunc(LPVOID param);

protected:
  virtual DWORD ReaderMemberFunc();
        DWORD getThreadId() { return m_ThreadId; };

  HANDLE  m_hThread;
  DWORD   m_ThreadId;
        BufferObject *bufferObj;
};

ReaderObject::ReaderObject()
{
  m_hThread = NULL;
  m_ThreadId = 0;
        BufferObject *bufferObj = new BufferObject();
}

void ReaderObject::StartThread(BufferObject & obj)
{
        bufferObj = &obj;
  m_hThread = (HANDLE)_beginthreadex(NULL,
    0,
    (PBEGINTHREADEX_THREADFUNC) ReaderObject::ReaderFunc,
    (LPVOID)this,
    0,
    (PBEGINTHREADEX_THREADID) &m_ThreadId );
  if (m_hThread) {
    cout << "Thread READER Id: " << getThreadId() << " launched" << endl;
  }
}



void ReaderObject::WaitForExit()
{
  WaitForSingleObject(m_hThread, INFINITE);

        DWORD threadExitCode;
```

```
    GetExitCodeThread(m_hThread, &threadExitCode);
    cout << "Thread READER Id: " << getThreadId()
         << " Exit Code = " << threadExitCode << endl;
CloseHandle(m_hThread);
}

//
// This is a static member function.  Unlike
// C static functions, you only place the static
// declaration on the function declaration in the
// class, not on its implementation.
//
// Static member functions have no "this" pointer,
// but do have access rights.
//
DWORD WINAPI ReaderObject::ReaderFunc(LPVOID param)
{
  // Use the param as the address of the object
  ReaderObject* pto = (ReaderObject*)param;
  // Call the member function. Since we have a
  // proper object pointer, even virtual functions
  // will be called properly.
  return pto->ReaderMemberFunc();
}

//
// This above function ThreadObject::ThreadFunc()
// calls this function after the thread starts up.
//
DWORD ReaderObject::ReaderMemberFunc()
{
  // Do something useful ...
      bufferObj->ReadFile(m_ThreadId);
return 0;
}
```

```
/*
 * buffer.h
 */

#define WIN32_LEAN_AND_MEAN


typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
  LPVOID lpThreadParameter
  );
typedef unsigned *PBEGINTHREADEX_THREADID;

//
// This ThreadObject is created by a thread
// that wants to start another thread. All
// public member functions except ThreadFunc()
// are called by that original thread. The
// virtual function ThreadMemberFunc() is
// the start of the new thread.
//
class BufferObject
{
public:
  BufferObject();
        void StartThread();
  void WaitForExit();
        int ReadFile(const DWORD &);
  int WriteFile(const DWORD &);
        static DWORD WINAPI BufferFunc(LPVOID param);

protected:
  virtual DWORD BufferMemberFunc();
        DWORD getThreadId() { return m_ThreadId; };

  HANDLE  m_hThread;
        HANDLE  hSemaphore;
        HANDLE  rSemaphore;
        HANDLE  hMutex;
  DWORD   m_ThreadId;
        int             iReader;
        int             iWriter;

};

BufferObject::BufferObject()
{
  m_hThread = NULL;
  m_ThreadId = 0;
        iReader = 0;
        iWriter = 0;

        hMutex = CreateMutex(NULL, FALSE, NULL);
        if (hMutex == NULL) {
                cout << "BUFFER Thread - error calling CreateMutex()" << endl;
                exit (0);
        }
        hSemaphore = CreateSemaphore(NULL,1, 1, NULL);

        if (hSemaphore == NULL) {

                cout << "BUFFER Thread - error calling CreateThreadSemaphore()" << endl;

                exit (0);

        }


void BufferObject::StartThread()
```

```
    m_hThread = (HANDLE)_beginthreadex(NULL,
      0,
      (PBEGINTHREADEX_THREADFUNC) BufferObject::BufferFunc,
      (LPVOID)this,
      0,
      (PBEGINTHREADEX_THREADID) &m_ThreadId );
    if (m_hThread) {
      cout << "Thread BUFFER Id: " << getThreadId() << " launched" << endl;
    }
  }

void BufferObject::WaitForExit()
{
  WaitForSingleObject(m_hThread, INFINITE);
        DWORD threadExitCode;

        GetExitCodeThread(m_hThread, &threadExitCode);
        cout << "Thread BUFFER Id: " << getThreadId()
                << " Exit Code = " << threadExitCode << endl;
  CloseHandle(hSemaphore);
        CloseHandle(hMutex);
        CloseHandle(m_hThread);

}

int BufferObject::ReadFile(const DWORD &dwThreadId)
{
        DWORD dwSemaphore;
        DWORD dwMutex;

        dwMutex = WaitForSingleObject(hMutex, INFINITE);
        cout << "Get Acquired the mutex: " << dwThreadId << endl;
        iReader++;
        if (iReader == 1) {
           dwSemaphore = WaitForSingleObject(hSemaphore, INFINITE);
                cout << "Get Acquire the Semaphore: " << dwThreadId << endl;
        }
        cout << "Released the mutex: " << dwThreadId << endl;
        ReleaseMutex(hMutex);

        cout << "Thread Id: " << dwThreadId << " reading file..." << endl;
        Sleep(1000);

        WaitForSingleObject(hMutex, INFINITE);
        cout << "Get Acquired the mutex: " << dwThreadId << endl;
        iReader--;
        if (iReader == 0) {
                cout << "Release the Semaphore: " << dwThreadId << endl;
                ReleaseSemaphore(hSemaphore, 1, NULL);
        }
        cout << "Released the mutex: " << dwThreadId << endl;
        ReleaseMutex(hMutex);

        return 0;



int BufferObject::WriteFile(const DWORD &dwThreadId)


        DWORD dwResult = WaitForSingleObject(hSemaphore, INFINITE);
        if (dwResult == WAIT_OBJECT_0) {
                cout << "Get Acquired the Semaphore: " << dwThreadId << endl;
                cout << "Thread Id: " << dwThreadId << " writing file..." << endl;
```

```
            Sleep(1000);
            cout << "Released the Semaphore: " << dwThreadId << endl;
            ReleaseSemaphore(hSemaphore, 1, NULL);
        }
        else {
                cout << "Error calling WaitForSingleObject()" << endl;
                return 0;
        }
        return 0;
}


//
// This is a static member function.  Unlike
// static functions, you only place the static
// declaration on the function declaration in the
// class, not on its implementation.
//
// Static member functions have no "this" pointer,
// but do have access rights.
//
DWORD WINAPI BufferObject::BufferFunc(LPVOID param)
{
  // Use the param as the address of the object
  BufferObject* pto = (BufferObject*)param;
  // Call the member function. Since we have a
  // proper object pointer, even virtual functions
  // will be called properly.
  return pto->BufferMemberFunc();
}


//
// This above function ThreadObject::ThreadFunc()
// calls this function after the thread starts up.
//
DWORD BufferObject::BufferMemberFunc()
{
  // Do something useful ...
        return 0;
}
```

APPENDIX C

READERS/WRITERS PROBLEM USING THE ASPECT-ORIENTED FRAMEWORK

```
/*
 * reader.h
 */

#define WIN32_LEAN_AND_MEAN

typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
  LPVOID lpThreadParameter
  );
typedef unsigned *PBEGINTHREADEX_THREADID;

//
// This ThreadObject is created by a thread
// that wants to start another thread. All
// public member functions except ThreadFunc()
// are called by that original thread. The
// virtual function ThreadMemberFunc() is
// the start of the new thread.
//
class ReaderObject
{
public:
  ReaderObject();
  void StartThread(ProxyObject &);
  void WaitForExit();

  static DWORD WINAPI ReaderFunc(LPVOID param);

protected:
  virtual DWORD ReaderMemberFunc();
        DWORD getThreadId() { return m_ThreadId; };

  HANDLE  m_hThread;
  DWORD   m_ThreadId;
        ProxyObject *proxyObj;
};

ReaderObject::ReaderObject()
{
  m_hThread = NULL;
  m_ThreadId = 0;

}

void ReaderObject::StartThread(ProxyObject & obj)
{
        proxyObj = &obj;
  m_hThread = (HANDLE)_beginthreadex(NULL,
    0,
    (PBEGINTHREADEX_THREADFUNC) ReaderObject::ReaderFunc,
    (LPVOID)this,
    0,
    (PBEGINTHREADEX_THREADID) &m_ThreadId );
  if (m_hThread) {
    cout << "Thread READER Id: " << getThreadId() << " launched" << endl;
  }
}
```

```
void ReaderObject::WaitForExit()
{
  WaitForSingleObject(m_hThread, INFINITE);

        DWORD threadExitCode;

        GetExitCodeThread(m_hThread, &threadExitCode);
        cout << "Thread READER Id: " << getThreadId()
                << " Exit Code = " << threadExitCode << endl;
  CloseHandle(m_hThread);
}


// This is a static member function.  Unlike
// C static functions, you only place the static
// declaration on the function declaration in the
// class, not on its implementation.
//
// Static member functions have no "this" pointer,
// but do have access rights.
//
DWORD WINAPI ReaderObject::ReaderFunc(LPVOID param)
{
  // Use the param as the address of the object
  ReaderObject* pto = (ReaderObject*)param;
  // Call the member function. Since we have a
  // proper object pointer, even virtual functions
  // will be called properly.
  return pto->ReaderMemberFunc();
}



// This above function ThreadObject::ThreadFunc()
// calls this function after the thread starts up.
//
DWORD ReaderObject::ReaderMemberFunc()
{
  // Do something useful ...
        Sleep(rand());
        proxyObj->Call("FileBuffer", "ReadFile", m_ThreadId);
  return 0;
}
```

```
/*
 *writer.h
 */

#define WIN32_LEAN_AND_MEAN

typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
 LPVOID lpThreadParameter
);
typedef unsigned *PBEGINTHREADEX_THREADID;

/*
 * This ThreadObject is created by a thread
 * that wants to start another thread. All
 * public member functions except ThreadFunc()
 * are called by that original thread. The
 * virtual function ThreadMemberFunc() is
 * the start of the new thread.
 */
class WriterObject
{
public:
        WriterObject();
 void StartThread(ProxyObject &);
 void WaitForExit();

 static DWORD WINAPI WriterFunc(LPVOID param);

protected:
 virtual DWORD WriterMemberFunc();
        DWORD getThreadId() {return m_ThreadId; };

 HANDLE  m_hThread;
        DWORD   m_ThreadId;
        ProxyObject *proxyObj;

};

WriterObject::WriterObject()
{
 m_hThread = NULL;
 m_ThreadId = 0;

}


void WriterObject::StartThread(ProxyObject & obj)
{
        proxyObj = &obj;
        m_hThread = (HANDLE)_beginthreadex(NULL,
 0,
 (PBEGINTHREADEX_THREADFUNC) WriterObject::WriterFunc,
 (LPVOID)this,
 0,
 (PBEGINTHREADEX_THREADID) &m_ThreadId );
        if (m_hThread) {
 cout << "Thread WRITER Id: " << getThreadId() << " launched" << endl;
}
}


void WriterObject::WaitForExit()
{
 WaitForSingleObject(m_hThread, INFINITE);
        DWORD threadExitCode;

        GetExitCodeThread(m_hThread, &threadExitCode);
```

```
        cout << "Thread WRITER Id: " << getThreadId()
                << " Exit Code = " << threadExitCode << endl;
    CloseHandle(m_hThread);


// This is a static member function.  Unlike
// static functions, you only place the static
// declaration on the function declaration in the
// class, not on its implementation.

// Static member functions have no "this" pointer,
// but do have access rights.

DWORD WINAPI WriterObject::WriterFunc(LPVOID param)

  // Use the param as the address of the object
  WriterObject* pto = (WriterObject*)param;
  // Call the member function. Since we have a
  // proper object pointer, even virtual functions
  // will be called properly.
  return pto->WriterMemberFunc();


DWORD WriterObject::WriterMemberFunc()

  // Do something useful ...
        Sleep(10);
        proxyObj->Call("FileBuffer", "WriteFile", m_ThreadId);
  return 0;



#include <string.h>
#include "AspectModerator.h"


class ProxyObject {
public:
        ProxyObject(){ };
        BOOL Call(const char *, const char * , const DWORD &);


private:
        AspectModerator amObject;


BOOL ProxyObject::Call(const char * sFuncObject, const char * sFuncName, const DWORD &obj)

        amObject.PointCut(sFuncObject ,sFuncName, obj);
        return TRUE;
```

```
/*
 * FileBuffer.h
 *
 */

class FileBuffer
{
public:
  FileBuffer() { };
      int ReadFile(const DWORD &);
  int WriteFile(const DWORD &);

protected:

  DWORD getThreadId() { return m_ThreadId; };
  DWORD m_ThreadId;

};

int FileBuffer::ReadFile(const DWORD &dwThreadId)
{
      Sleep(2000);
      cout << "Thread Id: " << dwThreadId << " reading file..." << endl;

      return 0;
}

int FileBuffer::WriteFile(const DWORD &dwThreadId)
{
      Sleep(2000);
      cout << "Thread Id: " << dwThreadId << " writing file..." << endl;
      return 0;
}
```

```
/*
 * SystemAbstractAspect.h
 */

class SystemAbstractAspect {
public:
        virtual void before(const char * sFuncName, const DWORD &obj) { };
        virtual void after(const char * sFuncName, const DWORD &obj) { };
private:
        virtual BOOL precondition(const char * sFuncName) = 0;
};


/*
 * SystemAbstractFactory.h
 */

#include <string.h>

class SystemAbstractAspectFactory {
public:
        virtual void precondition() = 0;
};

/*
 * SystemAspectFactory.h
 */

#include "SystemAspectFactory.h"
#include "SystemAspect.h"
#include "Synchronization.h"

class SystemAspectFactory : public SystemAbstractAspectFactory {
public:
        SystemAspectFactory();
        void attachImpl(SystemAbstractAspect *);
        virtual void precondition();
        virtual void before(const char * , const DWORD &);
        virtual void after(const char * , const DWORD &);

private:
        SystemAbstractAspect *pr_;
};

SystemAspectFactory::SystemAspectFactory()
{
        pr_ = 0;
}

void SystemAspectFactory::attachImpl(SystemAbstractAspect *pr)
{
        pr_ = pr;
}

void SystemAspectFactory::precondition()
{
        pr_->precondition();
}
void SystemAspectFactory::before(const char * sFuncName, const DWORD &obj)
{
        pr_->before(sFuncName, obj);
}

void SystemAspectFactory::after(const char * sFuncName, const DWORD &obj)
{
        pr_->after(sFuncName, obj);
}
```

```
/*
 *TracingAspect.h
 *
 */

#include "SystemAbstractAspect.h"

class TracingAspect : public SystemAbstractAspect {
public:
	TracingAspect();

	virtual void before(const char * , const DWORD &);
	virtual void after(const char * , const DWORD &);
private:

	DWORD	dwMutex;

	virtual BOOL precondition(const char * sFuncName);
};

TracingAspect::TracingAspect()
{
}


BOOL TracingAspect::precondition( const char * sFuncName)
{
	return TRUE;
}


void TracingAspect::before(const char * sFuncName, const DWORD &m_ThreadId)
{
	cout << "Executing " << sFuncName << endl;
}

void TracingAspect::after(const char * sFuncName, const DWORD &m_ThreadId)
{
	cout << "Exitting " << sFuncName << endl;
}
```

```
/*
 * Synchronization.h
 */

#include "SystemAbstractAspect.h"

class SynchronizationAspect : public SystemAbstractAspect {
public:
        SynchronizationAspect();

        virtual void before(const char * , const DWORD &);
        virtual void after(const char * , const DWORD &);
private:

        HANDLE  hSemaphore;
        HANDLE  wSemaphore;
        HANDLE  hMutex;

        DWORD   dwMutex;

        int             iReader;
        int             iWriter;
        int             i;
        int             iReaderWaiting;

        virtual BOOL precondition(const char * sFuncName);
};

SynchronizationAspect::SynchronizationAspect()
{

        iReader = 0;
        iWriter = 0;
        iReaderWaiting = 0;

        hMutex = CreateMutex(NULL, FALSE, NULL);
        if (hMutex == NULL) {
                cout << "Func Thread - error calling CreateMutex()" << endl;
                exit (0);
        }

        wSemaphore = CreateSemaphore(NULL,1, 1, NULL);
        if (wSemaphore == NULL) {
                cout << "Func Thread - error calling CreateThreadSemaphore()" << endl;
                exit (0);
        }

        hSemaphore = CreateSemaphore(NULL,1, 3, NULL);
        if (hSemaphore == NULL) {
                cout << "Func Thread - error calling CreateThreadSemaphore()" << endl;
                exit (0);
        }
}

BOOL SynchronizationAspect::precondition( const char * sFuncName)
{
        if ((char *) sFuncName == "ReadFile") {
                return (iWriter == 0);
        }
else
                if ((char *) sFuncName == "WriteFile") {
                return TRUE;
        }
        else return FALSE;
}

void SynchronizationAspect::before(const char * sFuncName, const DWORD &m_ThreadId)
```

```
{
        BOOL  bBlock = FALSE;

        if ((char *) sFuncName == "ReadFile") {
        cout << "READER before cross cut\n";
        dwMutex = WaitForSingleObject(hMutex, INFINITE);
        cout << "Reader Acquired the mutex: " << m_ThreadId << endl;

        switch (precondition(sFuncName)) {
        case FALSE:     cout << "Reader Released the mutex: " << m_ThreadId << endl;
                                iReaderWaiting++;
                                ReleaseMutex(hMutex);
                                WaitForSingleObject(hSemaphore, INFINITE);
                                WaitForSingleObject(hMutex, INFINITE);
                                iReader++;
                                cout << "READER Released the mutex: " << m_ThreadId << endl;
                                ReleaseMutex(hMutex);
                                break;
        case TRUE: iReader++;
                if (iReader == 1) {
                        cout << "READER Get Acquired the Semaphore: " << m_ThreadId << endl;
                        WaitForSingleObject(hSemaphore, INFINITE);
                }
                cout << "READER Released the mutex: " << m_ThreadId << endl;
                ReleaseMutex(hMutex);
                break;
        }

        }

        else if ((char *) sFuncName == "WriteFile") {
        cout << "WRITER before cross cut\n";
        dwMutex = WaitForSingleObject(hMutex, INFINITE);
        cout << "WRITER Get Acquired the mutex: " << m_ThreadId << endl;
        iWriter++;
        cout << "WRITER released the mutex: " << m_ThreadId << endl;
        ReleaseMutex(hMutex);
        cout << "WRITER Get Acquired the Semaphore: " << m_ThreadId << endl;
        WaitForSingleObject(hSemaphore, INFINITE);
        WaitForSingleObject(wSemaphore, INFINITE);
        }

}

void SynchronizationAspect::after(const char * sFuncName, const DWORD &m_ThreadId)
{

        if ((char *) sFuncName == "ReadFile") {
        cout << "READER after cross cut" << endl;
        WaitForSingleObject(hMutex, INFINITE);
        cout << "READER Get Acquired the mutex: " << m_ThreadId << endl;
        iReader--;
        if (iReader == 0) {
        cout << "READER Released the Semaphore: " << m_ThreadId << endl;
        ReleaseSemaphore(hSemaphore, 1, NULL);
        }
        cout << "READER Released the mutex: " << m_ThreadId << endl;
        ReleaseMutex(hMutex);
        }
        else
        {
                if ((char *) sFuncName == "WriteFile") {
                        cout << "WRITER after cross cut\n";
                        dwMutex = WaitForSingleObject(hMutex, INFINITE);
                        cout << "WRITER Get Acquired the mutex: " << m_ThreadId << endl;
                        iWriter--;
```

```
                cout << "WRITER released the Semaphore: " << m_ThreadId << endl;
                ReleaseSemaphore(wSemaphore, 1, NULL);
                // ReleaseSemaphore(hSemaphore, 1, NULL);
                if (iReaderWaiting > 0) {
                        cout << "WRITER released the Semaphore: READER\n";
                        for (i = 0; i < iReaderWaiting; i++)
                                ReleaseSemaphore(hSemaphore, 1, NULL);
                        iReaderWaiting = 0;
                        //ReleaseSemaphore(hSemaphore, iReaderWaiting, NULL);
                }
                else ReleaseSemaphore(hSemaphore, 1, NULL);
                cout << "WRITER released the mutex: " << m_ThreadId << endl;
                ReleaseMutex(hMutex);
        }
    }
}
```

```
/*
AspectModerator.h
*/

#include <string.h>
#include "SystemFactory.h"
#include "Sync1.h"
#include "Tracing.h"

class AspectModerator {
public:
        AspectModerator() { };
        BOOL PointCut(const char *, const char * , const DWORD &);

private:
        SystemAspectFactory sysAspectFactoryObj[2];
        FileBuffer functionalObject;
        SynchronizationAspectOne r;
        TracingAspect r1;
};

int AspectModerator::PointCut(const char * sFuncObject, const char * sFuncName, const DWORD &obj)
{
        if (strcmp(sFuncObject, "FileBuffer")==0) {

                if (strcmp(sFuncName,"WriteFile") == 0) {

                        sysAspectFactoryObj[0].attachImpl(&r);
                        sysAspectFactoryObj[0].before(sFuncName, obj);
                        functionalObject.WriteFile(obj);
                        sysAspectFactoryObj[0].attachImpl(&r);
                        sysAspectFactoryObj[0].after(sFuncName, obj);
                        sysAspectFactoryObj[0].attachImpl(&r1);
                        sysAspectFactoryObj[0].after(sFuncName, obj);
                }
                else
                if (strcmp(sFuncName,"ReadFile") == 0) {

                        sysAspectFactoryObj[0].attachImpl(&r);
                        sysAspectFactoryObj[0].before(sFuncName, obj);
                        functionalObject.ReadFile(obj);
                        sysAspectFactoryObj[0].attachImpl(&r);
                        sysAspectFactoryObj[0].after(sFuncName, obj);
                }
        }

        return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <windows.h>
#include <process.h>

#include "reader.h"
#include "writer.h"
//#include "buffer.h"

#define NUM_READER 20
#define NUM_WRITER 5

void main()
{
  ReaderObject reader[NUM_READER];
        WriterObject writer[NUM_WRITER];
        ProxyObject fileBuffer;


        for (int i = 0; i < NUM_WRITER; i++) {
                writer[i].StartThread(fileBuffer);
        }

        for (i = 0; i < NUM_READER; i++) {
                reader[i].StartThread(fileBuffer);
        }
        cin.get();

        for (i = 0; i < NUM_WRITER; i++) {
                writer[i].WaitForExit();
        }

        for (i = 0; i < NUM_READER; i++) {
                reader[i].WaitForExit();
        }

}
```

ภาคผนวก

Reprint ของบทความที่ได้รับการตีพิมพ์เผยแพร่ในการประชุมทางวิชาการจำนวน 6 บทความ

1. Paniti Netinant and Tzilla Elrad. "A Framework for Extensible and Adaptable System Software" in *Proceedings of the International Conference on Programming Languages and Compilers (PLC 2005)*, Las Vegas, Nevada, USA, June 2005.
2. Paniti Netinant. "Component + Aspect = an Extensible and Adaptable System Software" in *Proceedings of the International Conference on Software Engineering Research and Practices(SERP 2005)*, Las Vegas, Nevada, USA, June 2005.
3. Paniti Netinant. "Extensibility Aspect-Oriented Framework to Build Agent-Based System Software" in *Proceedings of the 15th International Conference on Software Engineering and Data Engineering (SEDE 2006)*, Los Angeles, California, USA, July 2006.
4. Paniti Netinant. "Extensible and Adaptable System Software" in *Proceedings of the International Conference on Programming Languages and Compilers (PLC 2006)*, Las Vegas, Nevada, USA, June 2006.
5. Paniti Netinant. "Supporting Separation of Concerns to Automation of Code Generation" in *Proceedings of the International Conference on Software Engineering Research and Practices (SERP 2006)*, Las Vegas, Nevada, USA, June 2006.
6. Paniti Netinant. "Building Agent-Based System Software Using Aspect-Oriented Framework" in *Proceedings of the 2006 Electrical Engineering/Electronics, Computer, Telecommunication, and Information Technology (ECTI) International Conference*, Thailand, May 2006.

PROCEEDINGS OF THE 2005 INTERNATIONAL CONFERENCE ON PROGRAMMING LANGUAGES AND COMPILERS

PLC'05

Editor

Hamid R. Arabnia

Associate Editors

Weichang Du
Christian Heinlein
Hassan Reza

Las Vegas, Nevada, USA
June 27-30, 2005

This volume contains papers presented at The 2005 International Conference on Programming Languages and Compilers (PLC'05). Their inclusion in this publication does not necessarily constitute endorsements by editors or by the publisher.

ISBN: 1-932415-75-0
Printed in the United States of America

CSREA Press
U. S. A.

Foreword

It gives us great pleasure to introduce this collection of papers to be presented at the 2005 International Conference on Programming Languages and Compilers (PLC'05), June 27 through 30, 2005, at Monte Carlo Resort, Las Vegas, Nevada, USA. The PLC'05 conference is co-sponsored and organized by CSREA; International Technology Institute (ITI); World Academy of Science for Information Technology (WAS-IT); as well as a number of institutions, computer science book publishers, users groups, newsgroups and a number of media co-sponsors (HPCwire, GRIDtoday, ...)

The program committee would like to thank all those who submitted papers for consideration. About 60% of the submissions were from outside the United States. Each submission was evaluated by two referees (except for papers that were submitted to chairs of sessions who were responsible for the evaluation of these papers.) The overall paper acceptance rate was about 37%.

We are very grateful to the many colleagues who helped in organizing the conference. In particular, we would like to thank the members of the PLC'05 Program Committee who we hope will offer their help again in organizing the next year's conference (PLC'06). The PLC'05 Program Committee members are:

- Prof. Hamid Abachi, Monash University, Australia (SERP, PLC)
- Prof. Hamid R. Arabnia (General Chair), University of Georgia, USA (SERP, PLC)
- Prof. Nadim Asif, Nat. Col. of Bus. Adm. & Ec. (NCBA& E), Pakistan (SERP)
- Dr. Punam Bedi, University of Delhi, Delhi, India (SERP)
- Dr. William Cheng-Chung Chu, TungHai University, Taiwan (SERP)
- Dr. Lawrence Chung, University of Texas at Dallas, USA (SERP)
- Dr. Constantinos Constantinides, Concordia University, Quebec, Canada (SERP)
- Dr. Heitor Augustus Xavier Costa, Universidade Federal de Lavras, Brazil (SERP)
- Dr. Juan J. Cuadrado-Gallego, University of Alcala, Madrid, Spain (SERP)
- Prof. Kevin Daimi, University of Detroit Mercy, Detroit, Michigan, USA (SERP)
- Dr. Sergiu Dascalu, University of Nevada, Reno, Nevada, USA (SERP)
- Dr. Charmaine DeLisser, University of Technology, Jamaica (SERP)
- Dr. Jing Dong, University of Texas at Dallas, Richardson, Texas, USA (SERP)
- Prof. Weichang Du, University of New Brunswick, New Brunswick, Canada (PLC)
- Dr. Emanuel Grant, University of North Dakota, USA (SERP)
- Dr. George A. Gravvanis, Democritus University of Thrace, Greece (SERP)
- Dr. Volker Gruhn, University of Leipzig, Germany (SERP)
- Dr. Jiang Guo, California State University Los Angeles, CA, USA (SERP)
- Dr. Fredrick C. Harris, Jr., University of Nevada Reno, Nevada, USA (SERP)
- Dr. Christian Heinlein, University of Ulm, Ulm, Germany (PLC)
- Dr. Mike Hinchey, NASA Goddard Space Flight Center, Greenbelt, MD, USA (SERP)
- Dr. Sumam Mary Idicula, Cochin University of Science & Technology, India (SERP)
- Dr. Carlos Juiz, Universitat de les Illes Balears, Spain (SERP)
- Dr. Osman Kandara, Southern University, Baton Roueg, Louisiana, USA (SERP)
- Prof. Hatsuhiko Kato, Shonan Institute of Technology, Japan (SERP, PLC)
- Prof. Fereydoun Kazemian, Rochester Institute of Technology, New York, USA (SERP)
- Dr. Anil Khatri, Johns Hopkins University, Maryland, USA (SERP)
- Dr. Dae-Kyoo Kim, Oakland University, Michigan, USA (SERP)

- Deborah Kobza, Director/CIO, Florida Inf. Tech. Center of Excellence, FL, USA (SERP)
- Dr. Cyril S. Ku, William Paterson University, New Jersey, USA (SERP)
- Dr. Kuan-Ching Li, Providence University, Shalu, Taichung, Taiwan (SERP)
- Prof. Prabhat K. Mahanti, University of New Brunswick, Canada (SERP)
- Dr. Johannes Mayer, University of Ulm, Germany (SERP)
- Dr. Marco T. Morazan, Seton Hall University, South Orange, New Jersey, USA (PLC)
- Prof. Youngsong Mun, Soongsil University, Korea (SERP)
- Dr. Paniti Netinant, Bangkok University, Bangkok, Thailand (SERP)
- Dr. Monica Nicolescu, University of Nevada, Reno, Nevada, USA (SERP)
- Prof. Michael Oudshoorn, Montana State University, Montana, USA (SERP, PLC)
- Dr. Jun Pang, INRIA, France (SERP)
- Dr. Lee Pike, NASA Langley Research Center, Hampton, Virginia, USA (SERP)
- Dr. Raghu Reddy, Colorado State University, Colorado, USA (SERP)
- Dr. Hassan Reza (Program Chair), University of North Dakota, ND, USA (SERP)
- Dr. Abdelhak Djamel Seriai, Ecole des Mines de Douai, Douai, France (SERP)
- Dr. H. Shrikumar, CTO of Ipsil Inc, USA (PLC)
- Dr. Roy Sterritt, University of Ulster at Jordanstown, Northern Ireland (SERP)
- Dr. Nary Subramanian, Hofstra University, Hempstead, New York, USA (SERP)
- Prof. Shantaram Vasikarla, American InterContinental University, LA, CA, USA (SERP)
- Brian Westphal (Student Member), University of Nevada, Reno, USA (SERP)
- Dr. Jon Whittle, NASA Ames Research Center, Moffett Fields, CA, USA (SERP)
- Prof. Baowen Xu, Southeast University, Nanjing, P. R. China (SERP)
- Dr. Haiping Xu, University of Massachusetts Dartmouth, Massachusetts, USA (SERP)
- Dr. Lu Yan, Turku Centre for Computer Science (TUCS), Turku, Finland (SERP)
- Dr. Wei Zhang, Southern Illinois University Carbondale, Illinois, USA (PLC)
- Dr. Mohammad Zulkernine, Queen's University, Ontario, Canada (SERP)

We would also like to thank the followings: UCMSS (Universal Conference Management Systems & Support, San Diego, California, USA) for managing all aspects of the conference; Dr. Tim Field of APC for managing the printing of the proceedings; and the staff of Monte Carlo Resort in Las Vegas for the professional service they provided.
Last but not least, we would like to thank PLC'05 Associate Editors, Drs. Weichang Du, Christian Heinlein, and Hassan Reza.

We present the proceedings of PLC'05.

Hamid R. Arabnia
PLC'05 Program Committee

Contents

SESSION: INTENSIONAL LANGUAGES, SYSTEMS & APPLICATIONS

The Lazy Evaluation of Infinitesimal Logic Expressions 3
Ruchi Agarwal, William W. Wadge

GIPSY – A Platform for the Investigation on Intensional Programming Languages 8
Joey Paquet, Aihua Wu

Toward JLucid, Lucid with Embedded Java Functions in the GIPSY 15
Peter Grogono, Serguei Mokhov, Joey Paquet

Objective Lucid – First Step in Object–Oriented Intensional Programming in the GIPSY 22
Serguei Mokhov, Joey Paquet

A Generic Framework for Migrating Demands in the GIPSY Demand–Driven Execution Engine 29
Emil Vassev, Joey Paquet

General Imperative Compiler Framework within the GIPSY 36
Serguei Mokhov, Joey Paquet

Object–Oriented Intensional Programming in the GIPSY: Preliminary Investigations 43
Aihua Wu, Joey Paquet

Lucx: Lucid Enriched with Context 48
Kaiyu Wan, Vasu Alagar, Joey Paquet

SESSION: LANGUAGE & COMPILERS FOR FUNCTIONAL PROGRAMMING

Towards Closureless Functional Languages 57
Marco, T Morazan

SequenceL – An Overview of a Simple Language 64
Daniel E. Cooke, J. Nelson Rushton

Toward Functionality Oriented Programming 71
Chengpu Wang

Components of Meta–Programming, Computer Analogies and Metaphors 78
David Dodds

SESSION: TECHNIQUES FOR MOBILE CODE, DISTRIBUTED SYSTEMS, & OO SYSTEMS

A Characterization of Traces in Java Programs 87
Borys Bradel, Tarek Abdelrahman

Cuckoo: a Language for Implementing Memory– and Thread–safe System Services 94
Richard West, Gary, T. Wong

Adding States into Object Types 101
Haitong Xu, Sheng Yu

A Programming System for Peer–to–Peer Computing 108
Weichang Du, Qian Jia

SESSION: PROGRAM ANALYSIS & COMPOSITION TECHNIQUES, AND TOOL SUPPORTS

Improved Passive Splitting 115
Keith Cooper, Jason Eckhardt

Null Values in Programming Languages 123
Christian Heinlein

Exploiting Syntactic Analysis for Lambda Lifting 130
Barbara Mucha, Marco, T Morazan

An Exceptional Programming Language 137
Mike Zastre, John Aycock

Towards Program Composing Assistants 142
Zorica Suvajdzin, Miroslav Hajdukovic

SESSION: LANGUAGE SUPPORT: SECURITY, SAFETY, EXCEPTIONAL HANDLING + GARBAGE COLLECTION

Inferring Java Security Policies Through Dynamic Sandboxing 151
Hajime Inoue

New Programming Language Concepts for Confidentiality 158

Christian Hummert, Gisela Menger

Exception Handling with Resumption: Design and Implementation in Java 165
Alexander Gruler, Christian Heinlein

Container Types for Automatic Garbage Collection in Hard Real–Time Computing 172
Kevin M Cleereman

Garbage Collection With a Large Address Space for Server Applications 179
Sergiy Kyrylkov, Darko Stefanovic

SESSION: EFFECTIVE TECHNIQUES FOR ADVANCED LANGUAGES FEATURES + LOOP OPTIMIZATIONS

Compilation Scheduling for the Java Virtual Machine 187
Robert Chun, Azeem, S Jiva

TUBE – Structure–Orientation in a Prototype–Based Programming Environment 194
Patrick Renner, Axel Rauschmayer

Micky: Methods With Implicit Calls 201
Bryan Crawley, Raphael Finkel

A Framework for Extensible and Adaptable System Software 207
Paniti Netinant, Tzilla Elrad

A Comparison of Z and UML: Two Case Studies 214
Bing Dong LI, M. H. Samadzadeh

SESSION: SUPPORT FOR UNANTICIPATED SOFTWARE EVOLUTION

Object and Access Evolution in Jarrah 223
Mark Evered

Component State Mapping for Runtime Evolution 230
Yves Vandewoude, Yolande Berbers

Targeting System Evolution by Explicit Modeling of Control Flows Using UML 2 Activity Charts 237
Stefan Sarstedt, Jens Kohlmeyer, Alexander Raschke, Matthias Schneiderhan

A Framework for Extensible and Adaptable System Software

Paniti Netinant[1,2] and Tzilla Elrad[2]

[1]Computer Science Department
Bangkok University
Bangkok, Thailand
paniti.n@bu.ac.th

[2]Computer Science Department
Illinois Institute of Technology
Chicago, IL, USA.
elrad@iit.edu

Abstract

Supporting separation of concerns in the design of operating systems can provide a number of benefits such as reusability, extensibility and reconfigurability. However, in order to maximize these benefits, such a support is difficult to accomplish. Some aspects in operating systems such as synchronization, scheduling, and fault tolerance cut across the basic functionalities of the system. In every layer these aspects might need to be modified. By separating the different aspects of operating system in every layer, we can provide a better generic design model of operating systems. Aspect-Oriented Programming is a paradigm proposal that aims at separating components and aspects from the early stages of the software life cycle, and combining them together at the implementation phase. However, aspect-oriented software engineering can be better supported if the underlying operating system is built based on this approach as well. Treating aspects, components, and layers in a two dimensional model is not adequate. Two-dimensional models lead to inflexibility, limit the possibilities for reuse, and make it hard to understand and modify. In this paper we discuss a language-neutral aspect-oriented design framework that can simplify system design by expressing it at a higher level of abstraction. Our work concentrates on how to maximize separation of aspects, components, and layers from each other. Our goal is to achieve a better design model for operating systems in terms of extensibility and adaptability.

Keywords— Adaptability, Aspect-Orientation, Framework, Extensibility, System Software.

1. System Software Design Issues

The commercialization of operating systems has resulted in their design gaining more importance. Operating systems are constantly extended for improvements as well as to support new features and hardware. In order to support this, reusability and adaptability [8] of system software during design is crucial. Extensibility provides the capability to either change current features or support new features. Reliability is the ability of the system to providing the correct service over a period of time. Stability is the capability of the system maintaining the correct service in every same state

Operating system design issues can be divided into hardware-oriented and software-oriented. Hardware–oriented issues include physical networks and communications protocol design, hardware measures, physical clock synchronization, storage, and system components. On the other hand, software-oriented issues include distributed algorithms, naming, resource allocation, distributed operating systems, system integration, reliability, tools and languages, real-time systems and performance measurement. The decisions involved in the design and implementation of operating systems address such issues as stability, reusability, adaptability, and reconfigurability.

2. Separation of Concerns

The principle of separation of concerns lies at the heart of software development as it introduces a number of benefits, originally addressed by [19, 7]. These include better understanding, modifiability, extensibility, debugging of the system, and better reuse of the concerns. Although these benefits have been well established, there is still no universally accepted methodology in order to guide a programmer to achieve it. The system designer has to consider how a number of aspects in the system can be captured, and how a separation of concerns [19, 12] will be addressed.

Functional decomposition has so far been achieved along one dimension, based on the underlying paradigm. In OOP, this dimension is a component hierarchy that includes methods, objects and classes. Current programming languages and techniques have been supportive to functional decomposition. Further, operating system design has also been based aligned with traditional functional decomposition techniques. However, no functional decomposition technique has yet managed to address a complete separation of concerns. Object Oriented Programming (OOP) seems to work well only if the problem can be described with relatively simple interface among objects. Unfortunately, this is not the case when we move from sequential programming to concurrent and distributed programming.

As distributed systems become larger, the interaction of their components is becoming more complex. This interaction may limit reuse and make it difficult to validate the design and correctness of software systems as well as force reengineering of these systems in order to meet future requirements. For complex software systems a solution to achieve separation of concerns without violating the benefits of OOP is still debatable and it constitutes a major research area. Certain properties of the systems do not localize well. Rather, they tend to cut across groups of functional components, making the system difficult to understand. The core complexity is that concurrent and distributed systems manifest over more than one dimension. Features such as scheduling, synchronization, fault tolerance, security, testing and verifications are all expressed in such a way that they tend to cut across groups of objects. This tangling of concerns [13], results in an increase of the dependencies between functional components that makes their source code difficult to understand, develop, and maintain. As a result, simple object interfaces are violated and the traditional OOP benefits no longer hold.

One current attempt to resolve this issue is the Aspect Oriented Software Architecture. To address this multi-dimensional structure of concurrent systems we distinguish between components and aspects. Aspects are defined as properties of a system that do not necessarily align with the system's functional components but tend to cut across groups of functional components, increasing their interdependencies, and thus affecting the quality of the software. Although not bound to OOP, Aspect-Oriented Programming (AOP) [10, 11, 13] is a paradigm proposal that retains the advantages of OOP and aims at achieving a better separation of concerns. AOP suggests that from the early stages of the software life cycle aspects should be addressed relatively separately from the functional components. As a result, aspectual decomposition manages to achieve a two-dimensional separation of concerns. At the implementation phase, aspects and components should be combined together, forming the overall system.

In this paper we address a number of operating system design issues based on a two-dimensional decomposition and in the context of the support provided by aspect-oriented frameworks [4, 5]. Our goals are to provide a cleaner separation of design concerns compared with what has so far been able to be supported by traditional approaches, better flexibility and higher reusability, as well as to provide a technique that would be practical to implement. Moreover, we want to demonstrate that the framework, based on an aspectual system decomposition, can be used effectively in this area as an alternative design approach that is also language independent.

3. System Components

The design of operating systems has been traditionally based either on the collective kernel structure (functional decomposition) or on the object-oriented model [22]. Examples of collective kernel structures include Mach [2] and Chorus [21]. Examples of object-oriented operating systems include Amoeba [23, 24] and Choice [3]. To our knowledge, there has so far been no proposal to address operating system design based on an aspect-oriented approach.

3.1 The Fault-Tolerance Cross-Cutting

As illustrating in figure 1 represents fault tolerance that cuts across the file system, the communications, and the time system.

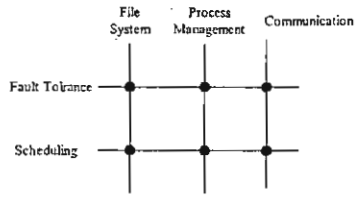


Figure 1. Fault tolerance and scheduling aspects cut across basic functional systems such as file sys-

...process management, and communication. An ...ation of this is given in Figure 2 where fault-...nt code is spread throughout the dataManager ...en in a programming language such as SR.

```
...e dataManager
...
...(...)
...(...)
...reToCommit()
...Manager(...)
...
...
...alize statusTable

...itor (ss) send failHandler()

...

...le for op; read/write, prepareToCommit...

...failHandler()
  destroy myresource()
...failHandler

...
  ...read(statusTable, sizeof(statusTable), statusTable);
  ...Manager.duUp(dmId);
...recovery
...dataManager
```

...ure 2. Fault-tolerant distributed software writ-... in programming language such as SR [1]. ...pted from [20])

...'s Distributed Computing Environment (DCE) ...] treats aspects in a monolithic manner residing ... object management layer. Further, object-...nted based systems like Amoeba [23, 16] and ...mice [3] treat aspects monolithically residing in ...ject management as well.

4. An Aspect-Oriented Framework

In [14, 15] AOP is viewed as a general frame-...k for separating the concerns in the system. ...kstra introduced the layered approach for the ...gn and implementation of operating systems ...h as the THE operating system [6] and the ...LTICS system [18]. The layered approach, ...nsisted of layers and components, has showed all ...e advantages of the modular design. Our observa-...n suggests that an aspect-oriented software ar-...tecture (AOSA) that uses aspect-oriented ...eworks could support designers and program-...rs in cleanly separating components and aspects ...m each other in different layers. AOSA can pro-...de a mechanism that would make it possible to ...tract and compose components and aspects to ...duce the overall system such as Aspect Modera-...r Framework [4]. We argue that a cross-cutting property of the system should not be seen within a two-dimensional model, and it should not be treated as a single monolithic aspect. Instead we propose the visualization of a three-dimensional model for system design. By adding the aspect dimension, we can capture aspects in the design. Three-dimensional model can simplify system design by expressing it at a higher level of abstraction

Our proposed framework is based on a three-dimensional system design that consists of components, aspects, and layers: 1) Components consist of the basic functionality modules of the system such as the file system, communication, and process management etc., 2) Aspects are cross-cutting entities, and they include fault tolerance, synchronization, scheduling, naming etc., 3) Layers consist of the components and aspects decomposed into a number of more manageable sub-problems. In general, lower layers deal with a far shorter time scale. The lower the layer, the closer the hardware is. The higher layer deals with interaction with the user.

By separating the different aspects of each component, we can separate components, aspects, and layers from each other (components from each other, aspects from each other, layers from each other, components from aspects, components in each layer, and aspects in each layer). It would thus be possible to abstract and compose them to produce the overall system. This would result in the clarification of interaction and increased understanding aspects of each component in the system. High-level of abstraction is easier to understand. Further, the reusability achieved by the higher level can use the lower level of the implementation not only to promote extensibility and refinement, but also to reduce cost and time in system development. A change in the implementation at a lower level would not result in a change at the higher level if the interface level has not been changed. Thus the design can achieve stability, consistency, and separation of concerns. An aspect might have multiple domains. Some aspect (scheduling, synchronization, naming, and fault tolerance e.g.) is scattered among many components in the system with varying policies, different mechanism, and possibly under different application.

To reduce the tangling of aspects in an operating system, each aspect should be considered and analyzed separately from the main functionality. For example, the aspect of scheduling in the file system can be considered in different domains in each layer. It would separate a policy from an as-

ject of each layer. Aspects would represent the general specifications needed to provide the abstraction. Further, a policy can be added or modified in each layer to a specific domain. This approach can support a high degree of reusability.

4 Architecture of the Framework

The proposed aspect-oriented framework for operating systems (Figure 3) is an extended model of the Aspect Moderator Framework [4, 5].

The overall framework architecture is divided into two frameworks based on two layers: a base framework on the low layer and an application framework on the upper layer. The Base Framework corresponds to the system layer. On the upper layer we may have more than one application frameworks.

The framework uses design patterns [9]. In this framework, aspects are created using the Abstract Factory and the Bridge patterns. The Abstract Factory would isolate aspects from implementation classes because the factory encapsulates the responsibility and the process of creating aspect objects. The class of concrete aspect appears only once in a functionality, where it's instantiated. The framework promotes consistency when an aspect is modified. The Bridge pattern avoids a permanent binding between an abstraction and its implementation. An example where this would be beneficial is when an implementation concern must be selected or switched at run-time. This way, different aspect abstractions and implementations can be combined and extended independently. This implementation is still useful when a change in the implementation of a class must not affect its existing components. As a result, a class need not be recompiled, but just re-linked. This approach supports polymorphism, and manages to avoid proliferation. Changing the implementation of an aspect abstraction should have no impact on functionality either. A smart-protection proxy controls access to the aspects and allows additional housekeeping tasks when an aspect is accessed.

In the application framework, the Adapter pattern allows the aspect factory to either convert the interface of an existing aspect (super aspect or aspects in the lower layers) into another interface functionality expect or to create a new aspect. Ideally, a new aspect should reuse an existing aspect to create new aspects, when it could be used. The upper layer can redefine existing aspects and override them.

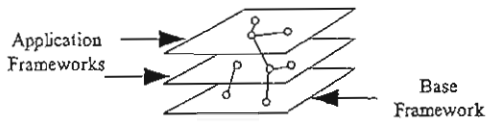


Figure 3. Aspect-Oriented Framework.

4.2 Base Framework

In this section we describe the execution flow in our design. We first describe the initialization and execution phases of the base framework. The application framework is discussed in a similar manner in the next section. We then discuss the adaptability issue of our framework. When the request arrives at the system, the smart-protection proxy will forward it to the AspectModerator object in order to identify whether this is a request to create an aspect or to invoke a method.

4.2.1 Initialization Phase

If there is an aspect creation request, the proxy will have to verify that this aspect does not already exist. Verification is achieved with communication with the AspectModerator object. Once verification is successful, the proxy will call the Aspectfactory to create the interface definition and the class definition of that aspect. The proxy will then register both definitions with the AspectModerator.

4.2.2 Invocation Phase

When a method is requested, the system evaluates some property constraints of that method, such as synchronization constraints. These constraints lie in an aspect object. The invocation of the constraints of the aspect will be executed by the AspectModerator. Once this is completed, the AspectModerator will return control to the proxy to resume the execution of the method of the main functionality whose invocation was requested by a client. This is done in a similar manner in both frameworks.

Once a request is identified a method invocation, the proxy will check whether an aspect that describes the method's constraints is already registered with the AspectModerator object. After a successful checking of the reference of the aspect, the AspectModerator will validate the constrains of the

invocation method. Then, the AspectModerator will activate the method of the aspect object and return control to the proxy.

4.3 Application Framework

As with the base level, the application framework is discussed in the context of its initialization and method invocation phases.

4.3.1 Initialization Phase

When the smart-protection proxy receives a request, it will recognize whether the request refers to an aspect creation, or a method invocation. The proxy will check the aspect moderator if there is an aspect registered in the AspectModerator, and which aspects in the lower layer are included in the application layer. If a registration entry of the aspect does not exist, the proxy will call the AspectFactory to create the interface and the class of that aspect. The proxy will then register both definitions with the AspectModerator. The new aspect can be included the lower layer aspect to build a new aspect. Then the proxy will register the reference of that aspect in the AspectModerator.

4.3.2 Invocation Phase

During method invocation, the proxy will look up the register at the AspectModerator of that layer to see whether a lower-layer reference exists for that aspect. If there is no reference to that aspect, the proxy will look up the lower layer. Should the requested aspect has not been registered in neither layer, the AspectModerator will return an error to proxy. After a successful checking of the reference of the aspect, the AspectModerator will validate the constrains of the method that is invoked. The AspectModerator will then activate the method, and return control to the proxy.

4.4 Adaptability of Framework

The general architecture of the framework promotes reusability (the upper layer can reuse aspects from the lower layer), extensibility, and ensures adaptability of aspects and components because both are designed and implemented relatively separately from each other. Aspects in the application framework can be extended and redefined by aspects provided by the layer to meet new requirements. A new aspect can be added in both system layer and application layer without interfering with aspects or components in other layer. The AspectModerator in both frameworks need not be modified when a new aspect is introduced.

5. Discussion

The Aspect-Oriented Design Framework is a three-dimensional model that consists of a collection of aspects, and which can provide an abstraction in the operating system to support a number of components in the upper levels. Components form the main functionality of the operating system. Layers can be divided into lower, intermediate, and upper level. The lower level represents the operating system that provides reusable primitives for the intermediate and upper levels. The intermediate level corresponds to the system programming or interface definition. The upper level corresponds to application and programming level.

5.1 Reusability, Stability and Consistency

The abstraction of an aspect in the lower level provides transparency. The upper aspects or components can use the lower aspects or components without knowing the internal details of how the lower aspects or components are implemented. Information hiding promotes either component or aspect modifiability and simplifies the perception of the upper level. The upper level component or application can use the abstraction of aspect and component in the lower level without knowing the internal details of how the lower level aspect is constructed. If the lower aspect is changed (to improve performance or to add new features, for example), provided the aspect interface (intermediate level) remains constant, the upper level aspect need not change. This approach could result in better modifiability of the system.

5.2 Extensibility

Software reusability not only can save time in program development, but it can also avoid unnecessary proliferation of functions. By reusing of proven and debugged high quality software, it will reduce problems after a system becomes operational. Polymorphism enables us to provide a generality of aspect to handle a wide variety of policies. It also makes it easy to add new capabilities to

an aspect. It will help to deal with complexity and redundancy in the system, and could be particularly effective for implementing layered software systems. When fundamental aspects of the system such as scheduling, synchronization, and fault tolerance are created, we can define them as a super-aspect. When creating a new aspect (to change or add a policy for example) we can designate that a new aspect is to either inherit from or override its super-aspect rather than being re-written. This new aspect is refereed to as a sub-aspect.

5.3 Adaptability

An aspect abstract is a super-aspect provided by the system and it can be reused and redefined by sub-aspects in the upper levels. The sole purpose of an abstract aspect is to provide an appropriate super-aspect from which other aspects could inherit, override or redefine implementation. Process management can redefine scheduling by round robin. Communication component might need FCFS. The file system would then need to use the same policy as process management. For example, a database application does a scan of one portion of its memory, while doing random access to another portion. A scheduling of a file system implementation using LRU replacement policy will perform poorly on the scanned memory. A scheduling of a file system should be capable to reconfigure to an appropriate policy for a better performance.

5.4 Architectural Independence

An aspect provides an independent architecture because each aspect is not a modular unit such as a procedures or an object. System design should begin by focusing attention on the problem to be solved, postponing considerations of architecture and language constructs. At the implementation level, aspects can be modeled in abstractions like classes in Object-Oriented Programming or aspects in Aspect-Oriented Programming.

6. Conclusion

Operating system design should not be seen as a two-dimensional model of layers and components that includes single monolithic aspects. In this paper, we stressed the importance of the complete separation of concerns within the context of an aspect-oriented framework and we discussed how this technique could provide an alternative to operating system design and implementation. In this paper we described an aspect-oriented framework where both functional components and aspects are designed relatively separately from each other. This separation of concerns allows for reusability and adaptability. Our work concentrates on the decomposition of aspects and components in software systems and our goal is to achieve an improved separation of concerns in both the design and the implementation. Our design framework provides an adaptable model that allows for an open language where new aspects can be manageable and added in both applications and operating systems easier. The interaction of newly added aspects is specified by a contract that binds a new aspect to the rest of the system rather than having to re-engineer the whole system. The framework approach is promising, as it seems to be able to address a large number of system aspects and applications. The advantage of decomposing of functional components and aspect in every layer is to promote reusability, adaptability and manageability of both components and aspects in operating systems easier. Refinement of aspects of the system can further be achieved by polymorphism.

7. References

[1] Andrew, G., Olsson, R., *The SR Programming Language:Concurrency in Practice,* Benjamin/Cumming, 1993.

[2] Acetta, M., Barron R., Bolosky W., Golub G., Rashid R., Tevanian A., and Young M. Mach: A New Kernel Foundation for UNIX Development. *Proceedings of the Summer USENIX Conference*, p.93-113, June 1986.

[3] Campbell, R. H., G. N. Johnston, P. W. Madany, and V. F. Russo. Principles of Object-Oriented Operating System Design, *Technical Report R89-1510*, University of Illinois, April 1989.

[4] Constantinos Constantinides, Atef Bader, Tzilla Elrad. A Framework to Adress a Two-Dimensional Separation of Concens, *OOPSLA Workshop on Multidimentional Separation of Concerns,* 1999.

[5] Constantinos A. Constantinides, Atef Bader, Tzilla Elrad, Mohamed E. Fayad, Paniti Netinant. Designing an Aspect-Oriented Framework in an Object-Oriented Environment, *ACM Computing Surveys*, March 2000.

PROCEEDINGS OF THE 2005 INTERNATIONAL
CONFERENCE ON SOFTWARE ENGINEERING
RESEARCH AND PRACTICE

SERP'05

Volume I

Editors

Hamid R. Arabnia
Hassan Reza

Associate Editors

Lawrence Chung, Juan J. Cuadrado-Gallego
Sergiu Dascalu, Emanuel Grant,
Frederick C. Harris Jr., Michael Hinchey
Deborah Kobza, Youngsong Mun, Roy Sterritt
Nary Subramanian

Las Vegas, Nevada, USA
June 27-29, 2005

This volume contains papers presented at The 2005 International Conference on Software Engineering Research and Practice (SERP'05). Their inclusion in this publication does not necessarily constitute endorsements by editors or by the publisher.

ISBN: 1-932415-49-1, 1-932415-50-5 (1-932415-51-3)
Printed in the United States of America

CSREA Press
U. S. A.

Foreword

It gives us great pleasure to introduce this collection of papers to be presented at the 2005 International Conference on Software Engineering Research and Practice (SERP'05), June 27 through 30, 2005, at Monte Carlo Resort, Las Vegas, Nevada, USA. The SERP'05 conference is co-sponsored and organized by CSREA; International Technology Institute (ITI); World Academy of Science for Information Technology (WAS-IT); as well as a number of institutions, computer science book publishers, users groups, newsgroups and a number of media co-sponsors (HPCwire, GRIDtoday, ...)

The program committee would like to thank all those who submitted papers for consideration. About 50% of the submissions were from outside the United States. Each submission was evaluated by two referees (except for papers that were submitted to chairs of sessions who were responsible for the evaluation of these papers.) The overall paper acceptance rate was about 36%.

We are very grateful to the many colleagues who helped in organizing the conference. In particular, we would like to thank the members of the SERP'05 Program Committee who we hope will offer their help again in organizing the next year's conference (SERP'06). The SERP'05 Program Committee members are:

- Prof. Hamid Abachi, Monash University, Australia (SERP, PLC)
- Prof. Hamid R. Arabnia (General Chair), University of Georgia, USA (SERP, PLC)
- Prof. Nadim Asif, Nat. Col. of Bus. Adm. & Ec. (NCBA& E), Pakistan (SERP)
- Dr. Punam Bedi, University of Delhi, Delhi, India (SERP)
- Dr. William Cheng-Chung Chu, TungHai University, Taiwan (SERP)
- Dr. Lawrence Chung, University of Texas at Dallas, USA (SERP)
- Dr. Constantinos Constantinides, Concordia University, Quebec, Canada (SERP)
- Dr. Heitor Augustus Xavier Costa, Universidade Federal de Lavras, Brazil (SERP)
- Dr. Juan J. Cuadrado-Gallego, University of Alcala, Madrid, Spain (SERP)
- Prof. Kevin Daimi, University of Detroit Mercy, Detroit, Michigan, USA (SERP)
- Dr. Sergiu Dascalu, University of Nevada, Reno, Nevada, USA (SERP)
- Dr. Charmaine DeLisser, University of Technology, Jamaica (SERP)
- Dr. Jing Dong, University of Texas at Dallas, Richardson, Texas, USA (SERP)
- Prof. Weichang Du, University of New Brunswick, New Brunswick, Canada (PLC)
- Dr. Emanuel Grant, University of North Dakota, USA (SERP)
- Dr. George A. Gravvanis, Democritus University of Thrace, Greece (SERP)
- Dr. Volker Gruhn, University of Leipzig, Germany (SERP)
- Dr. Jiang Guo, California State University Los Angeles, CA, USA (SERP)
- Dr. Fredrick C. Harris, Jr., University of Nevada Reno, Nevada, USA (SERP)
- Dr. Christian Heinlein, University of Ulm, Ulm, Germany (PLC)
- Dr. Mike Hinchey, NASA Goddard Space Flight Center, Greenbelt, MD, USA (SERP)
- Dr. Sumam Mary Idicula, Cochin University of Science & Technology, India (SERP)
- Dr. Carlos Juiz, Universitat de les Illes Balears, Spain (SERP)
- Dr. Osman Kandara, Southern University, Baton Roueg, Louisiana, USA (SERP)
- Prof. Hatsuhiko Kato, Shonan Institute of Technology, Japan (SERP, PLC)
- Prof. Fereydoun Kazemian, Rochester Institute of Technology, New York, USA (SERP)
- Dr. Anil Khatri, Johns Hopkins University, Maryland, USA (SERP)
- Dr. Dae-Kyoo Kim, Oakland University, Michigan, USA (SERP)

- Deborah Kobza, Director/CIO, Florida Inf. Tech. Center of Excellence, FL, USA (SERP)
- Dr. Cyril S. Ku, William Paterson University, New Jersey, USA (SERP)
- Dr. Kuan-Ching Li, Providence University, Shalu, Taichung, Taiwan (SERP)
- Prof. Prabhat K. Mahanti, University of New Brunswick, Canada (SERP)
- Dr. Johannes Mayer, University of Ulm, Germany (SERP)
- Dr. Marco T. Morazan, Seton Hall University, South Orange, New Jersey, USA (PLC)
- Prof. Youngsong Mun, Soongsil University, Korea (SERP)
- Dr. Paniti Netinant, Bangkok University, Bangkok, Thailand (SERP)
- Dr. Monica Nicolescu, University of Nevada, Reno, Nevada, USA (SERP)
- Prof. Michael Oudshoorn, Montana State University, Montana, USA (SERP, PLC)
- Dr. Jun Pang, INRIA, France (SERP)
- Dr. Lee Pike, NASA Langley Research Center, Hampton, Virginia, USA (SERP)
- Dr. Raghu Reddy, Colorado State University, Colorado, USA (SERP)
- Dr. Hassan Reza (Program Chair), University of North Dakota, ND, USA (SERP)
- Dr. Abdelhak Djamel Seriai, Ecole des Mines de Douai, Douai, France (SERP)
- Dr. H. Shrikumar, CTO of Ipsil Inc, USA (PLC)
- Dr. Roy Sterritt, University of Ulster at Jordanstown, Northern Ireland (SERP)
- Dr. Nary Subramanian, Hofstra University, Hempstead, New York, USA (SERP)
- Prof. Shantaram Vasikarla, American InterContinental University, LA, CA, USA (SERP)
- Brian Westphal (Student Member), University of Nevada, Reno, USA (SERP)
- Dr. Jon Whittle, NASA Ames Research Center, Moffett Fields, CA, USA (SERP)
- Prof. Baowen Xu, Southeast University, Nanjing, P. R. China (SERP)
- Dr. Haiping Xu, University of Massachusetts Dartmouth, Massachusetts, USA (SERP)
- Dr. Lu Yan, Turku Centre for Computer Science (TUCS), Turku, Finland (SERP)
- Dr. Wei Zhang, Southern Illinois University Carbondale, Illinois, USA (PLC)
- Dr. Mohammad Zulkernine, Queen's University, Ontario, Canada (SERP)

We would also like to thank the followings: UCMSS (Universal Conference Management Systems & Support, San Diego, California, USA) for managing all aspects of the conference; Dr. Tim Field of APC for managing the printing of the proceedings; and the staff of Monte Carlo Resort in Las Vegas for the professional service they provided.
Last but not least, we would like to thank SERP'05 Associate Editors, Drs. Lawrence Chung, Juan J. Cuadrado-Gallego, Sergiu Dascalu, Emanuel Grant, Frederick C. Harris, Jr., Michael Hinchey, Deborah Kobza, Youngsong Mun, Roy Sterritt, and Nary Subramanian.

We present the proceedings of SERP'05.

Hamid R. Arabnia and Hassan Reza
SERP'05 Program Committee

Contents

SESSION: AUTONOMIC & AUTONOMOUS SPACE EXPLORATION SYSTEMS – A & A–SES'05

SPAACE :: Self– Properties for an Autonomous &Autonomic Computing Environment 3
Roy Sterritt, Mike Hinchey

Progressive Autonomy – An Incremental Agent–based Approach 9
Walt Truszkowski, Christopher Rouff, Sidney Bailin, Mike Rilee

Specification and Implementation of Autonomic Large–Scale System Behaviors Using Domain Specific Modeling Language Tools 16
Shikha Ahuja, Shwetta Shetty, Sandeep Neema, Di Yao, Steve Nordstrom, Ted Bapty

Experimenting with an Evolving Ground/Space–based Software Architecture to Enable Sensor Webs 23
Daniel Mandl, Stuart Frye

A Model–Based Approach to Controlling the ST–5 Constellation Lights–Out Using the GMSEC Message Bus and Simulink 29
Jason Stanley, Robert Shendock, Kenneth J. Witt, Daniel Mandl

Autonomous and Autonomic Swarms 36
Michael Hinchey, James Rash, Walter Truszkowski, Christopher Rouff, Roy Sterritt

SESSION: INT'L WORKSHOP ON SYSTEM/SOFTWARE ARCHITECTURES – IWSSA'05

Performance Assessment of Architectural Options on Intelligent Distributed Systems 45
Carlos Juiz, Joachim Zottl, GÃ¼nter Haring, Ramon Puigjaner

A Framework for Reuse and Parallelization of Large–Scale Scientific Simulation Code 52
Manolo Sherrill, Roberto Mancini, Frederick Harris, Sergiu Dascalu

A Software Architecture Intended to Design High Quality Groupware Applications 59
José Luis Garrido, Patricia Paderewski, María Luisa Rodríguez, Miguel J. Hornos, Manuel Noguera

Self*– Properties in NASA Mission 66
Roy Sterritt, Christopher Rouff, James Rash, Walter Truszkowski, Michael Hinchey

V–FIRE: Virtual Fire in Realistic Environments 73

Frederick Harris, Michael Penick, Grant Kelly, Juan Quiroz, Sergiu Dascalu, Brian Westphal

Scope Equivalence of Concurrent Systems Based on Bipartite Directed Acyclic Graph 80
Masaki Murakami

Interface Descriptions for Enterprise Architecture 87
Aditya Garg, Rick Kazman

Detection of Anomalies in a Software Architectural Style with Connectors: Position Paper 94
Michael Shin, Yan Xu

Specification and Performance Metrics for Parallel Programs 101
Brian d'Auriol, Juan Ulloa

Supporting the Development of Adaptable and Secure Software Systems: An NFR Approach 108
Nary Subramanian, Lawrence Chung

Evaluating Off–The–Shelf Architectural Components 115
Kendra Cooper, Lawrence Chung, Weimin Ma

Privacy Aware Identity Information Sharing Protocol 122
Taesung Kim, Jong hyuk Roh, Seung–Hyun Kim

Giving Feedback on MASCOTime Simulation Results 126
Pere P. Sancho, Carlos Juiz, Ramon Puigjaner

SESSION: SOFTWARE DESIGN

Using Semantic Metrics to Assess Consistency between Design and Implementation of Software 135
Cara E. Stein

The Effect of Object–Oriented Data Structure Design and Implementation on Lifecycle Effort: A Case Study 142
Jack K. Horner

Implementing Multiple Priorities in a Publish–Subscribe System for Netcentric Applications 149
Margaret McMahon

Applying Design Patterns in Distributing a Genetic Algorithm Application 154
Nick Burns, Mike Bradley, Mei–Ling Liu

UML–based Beowulf Cluster Availability Modeling 161
Hertong Song, Chokchai Leangsuksun, Raja Nassar, Yudan Liu, Christian Engelmann, Stephen Scott

Meta–Model Search: Using XPath to Search Domain–Specific Models 168
Rajesh Sudarsan, Jeff Gray

UML 2.0 Consistency Rules Identification 175
Jean–Pierre Seuma Vidal, Hugues Malgouyres, Gilles Motet

Executable Visual Software Modeling – The ZOOM Approach 182
Xiaoping Jia, Adam Steele, Lizhang Qin, Hongming Liu, Chris Jones

Metainterfaces Support Structural and Object–Oriented Software Composition 189
Enn Tyugu

SESSION: SOFTWARE RELIABILITY, ASSURANCE, SECURITY & PROJECT MANAGEMENT

A Survey of Software Reliability Models and an Application of the Bayesian Belief Networks Model 195
Qiaolan Wan, Mansur H. Samadzadeh

Discrete Time Modelling In Software Reliability – A Unified Approach 202
Nazar Sarhan, Omar Shatnawi

Novel Obfuscation Algorithms for Software Security 209
Suma Venkatesh, Levent Ertaul

An Attack Packet Simulator for Performance Test of Information Security Systems 216
Wooyoung Soh, Junsang Jeon, Younseo Jeong

Strengthening Software Integrity through Privacy and Security Requirements Modelling 223
Matthew Nicolas Kreeger

Electronic Voting Systems Security Requirements Engineering 230
Clarence Wilson, Kevin Daimi

SESSION: FORMAL METHODS

GUI State–based Accessibility Control in Hierarchical State Machines 239
Mingtian Ni, Stephen, E Reichenbach

Towards Arguing the Cost–effectiveness of Coloured Petri Nets 246
Jens Bæk Jørgensen

Graph Theory in the Control Flow Analysis of the Large Time Critical Applications 253
Sergej Alekseev, Günther Stiege

Generation of OCL Constraints from B Abstract Machines 260
Jean–Christophe Voisinet, Bruno Tatibouet, Isabelle Jacques

Verification of Workflow Authorization Reasonability 267
Ouyangyu Yu, Liu Yu–shu

On a GUI based Editor for the Z Notation 273
Hiroshi Ishikawa

SESSION: SOFTWARE TESTING

Pseudo–Random System Testing: Coverage Estimation and Enhancement 283
Ali, Y Duale, Theodore, J Bohizic, Dennis, W Wittig

A Complete Automation of Unit Testing for Java Programs 290
Yoonsik Cheon, Myoung Kim, Ashaveena Perumandla

Analysis of Open Source Defect Tracking Tools for Use in Defect Estimation 296
Dileep Potnuri, Catherine V. Stringfellow

A Low Budget Approach to Distributed Automated Black–Box Testing 302
Andreas Boklund, Christer Selvefors

A Process Model for Development and Utilization of Reusable Test Assets 309
Annukka Mäntyniemi, Pekka Mäki–Asiala, Matti Kärki

Meta–Modeling Approach to Tool Support for Model Transformation to Validate Dynamic Behavior of Systems 316
Michael Shin, Marta Calderon

GUI Test Case Generation from UML 323
Yachai Limpiyakorn, Petnamkang Wongsuttipakorn

Using Architectural Modeling for Integration Testing 330
Hassan Reza, Emanuel Grant

Test Case Prioritization for GUI Testing 338
Yachai Limpiyakorn, Pisak Kurusathian

SESSION: SOFTWARE ARCHITECTURE & UML/MDA I

Modeling QoS through Architectural Reflection 347
Francesca Arcelli, Claudia Raibulet, Francesco Tisato

Model Reuse in MDA 354
Bouzitouna Salim, Gervais Marie–Pierre, Blanc Xavier

A Transformation Approach for Modeling and Analysis of Complex UML Statecharts: A Case Study 361
Zhaoxia Hu, Sol Shatz

Software Architecture Evolution: Description and Management Process 368
Nassima Sadou, Mourad Oussalah, Dalila Tamzalit

Maturity of the MDA Tool–assisted Development Process using Business Archetypes: a Case Study 375
Eric Lefebvre, Blanca Gil

JfelX: A Dynamic Model Driven Architecture 382
Abhiram Gandhe, Puneet Agarwal, Gautam Shroff

A Domain Composition Approach 389
Jacky Estublier, Anca Daniela Ionita, German Vega

A New Approach to Combine Models and Code in Model Driven Development 396
Stefan Sarstedt, Jens Kohlmeyer, Alexander Raschke, Matthias Schneiderhan

SESSION: SOFTWARE REUSE & ASPECT–ORIENTED

Component + Aspect = an Extensible and Adaptable System Software 403
Paniti Netinant

Integration of Wrapper Preprocessor into C/C++ IDE for Implementing Reusable Components 408
Walter Fortner, Hisham Haddad

SESSION: SOFTWARE EDUCATION

A Cultural Shift in Teaching CS Programming Courses and Improving Software Quality 417
Mohammed Gomaa, Akram Salah, Syed Rahman

SESSION: REQUIREMENT ENGINEERING

The Views of Quality for the Requirements Document 427
Bernard Wong

Software Requirements Phase for a Resource Utilization and Scheduling Tool 435
Jayathi Raghavan, Massood Towhidnejad

Research on Software Requirement Analysis Method Based on Five–Key Elements Arrange 442
Wei Fu, Guoqiang Cai, Limin Jia, Yangdong Ye, Ye Zhang

A Fuzzy Logic–based Approach for Requirements Elicitation Techniques Selection 448
Yirsaw Ayalew, Semahegn Abebe

SESSION: ADVANCED MODELS IN SOFTWARE PROJECTS DEVELOPMENT

Evaluation of the Legal Certainty in the IT 457
Ricardo J. Rejas–Muslera, José A. Gutierrez–de–Mesa

A Study of the Relationship between Usability and Test Cases Precedence Based on a Formal Model for Activity Diagrams 462
Pedro J. Lara Bercial, Juan José Escribano Otero, Luis Fernández Sanz, José Ramón Hilera Gonzalez

Effort Estimation in Agile Software Development: A Method and a Case Study 470
Fernando Machado, Luis Joyanes

Optimizing Software Construction 476
Dario Paciarelli, Marco Pranzo, Juan J. Cuadrado–Gallego, María D. Rodriguez–Moreno

On the Sizing of Knowledge Management Activities and its Relationship to Supporting Technology 483
Miltiadis Lytras, Miguel–Angel Sicilia, Danai Tsotra

SESSION: PROGRAM ANALYSIS & EXTREME PROGRAMMING

Analyzing Static Structure of Large Software Systems 491
James W. Fawcett, Murat K. Gungor, Arun V. Iyer

JavaContexts: A Java Based Programming Language for the Development of Highly Reusable Software Applications 497
Waldemar Wieczerzycki

Meta Code Pattern and Its Refinement 504
Jian Liu, Farokh Bastani, I-Ling Yen

Specifying and Checking Method Call Sequences in JML 511
Yoonsik Cheon, Ashaveena Perumandla

An Approach to Transforming Parallel Function Specification into Java Program Framework 517
Tong Li, Hongji Yang, Baowen Xu, Liang Shi

Metrics for Multithreaded Java Program Verification 524
Ahmed Salem, Varun Sharam

Experiments on Mutual Dependence between Class Analysis and Exception Analysis 529
Jang-Wu Jo, Keehang Kwon

TXP (Traditionally Extreme Programming) 534
Tarek Alameldin, Vishal Saberwal

SESSION: USABILITY & WEB ENGINEERING

Reconfigurable Interfaces: A Proposal for Universal Usability Interfaces 543
Mohammed Gomaa, Akram Salah, Syed Rahman

Version Control in Online Software Repositories 550
E. Rowland Watkins, Denis A. Nicole

Modeling Services and Web Services: Application of ModelBus 557
Xavier Blanc, Prawee Sriplakich, Marie-Pierre Gervais

Software Engineering Methodology for E-Commerce Estimation 564
A. Hameed Al-Elaiwi

Prioritizing Web Usability Attributes Using Intuitionistic Fuzzy Sets 570
Punam Bedi, Hema Banati

SESSION: EMPIRICAL/CASE STUDIES & TOOLS

Application of Software Engineering Fundamentals: A Hands on Experience 579
Cihan Varol, Coskun Bayrak, Robert Ludwig

Implementation of Network Event Audit Module Using Data Mining Method 584
Wooyoung Soh, Seakjae Han, Wankyoung Kim, Suksoo Kim, Namsun Choi

Apollo 11 Revisited – An Example of Problem–Based Learning 590
Josepr, R. Bumblis

Introducing Contract–Based Programming in Industry – a Case Study 596
Martin Blom, Eivind Nordby, Anna Brunstrom

Advanced ISO/IEC 12207 for SOC 602
Kyunghee Lee, Hyemin Kim, In–Sup Paik, Minkoo Kim

The Kaburobo Contest 606
Nachi Ueno, Motohiro Shamoto, Koichi Kato, Yoichi Muraoka

A Co–Work Tool for Visual Programs 611
Yong–Yun Cho, Chae–Woo Yoo

Updating Scientific Legacy Systems to Bridge the Digital Divide: A Case Study 618
H. Keith Edwards, Robert Puckett, Don Thomas

SESSION: PERFORMANCE MODELING, ANALYSIS, & RISK ASSESSMENTS

Modeling the performance of the Web service platform using Layered Queueing Networks 627
Sofie Van Hoecke, Tom Verdickt, Filip De Turck, Bart Dhoedt, Piet Demeester

Dynamic Informational System for Control and Monitoring the Tritium Removal Pilot Plant with Data Transfer and Process Analyses 634
Carmen Maria Retevoi, Iuliana Stefan, Ovidui Balteanu, Liviu Stefan

Software Development Risk Model 640
James W. Fawcett, Murat K. Gungor

SESSION: SOFTWARE COST ESTIMATION

A Software Cost Estimation Model for Product Line Engineering: SoCoEMo–PLE 649
Sana Ben Abdallah Ben Lamine, Lamia Labed Jilani, Henda Hajjami Ben Ghezala

A Requirement–Based Project Estimation Approach 656
Ching–Pao Chang, Ching–seh Wu, Chih–Ping Chu, Jia–Lyn Lv

A Software Complexity Measurement Technique for Object–Oriented Reverse Engineering 663
Jongwan Kim, Chong–Sun Hwang, Joo Ho Choi

Demonic Fixed Point of Semantics Function 670
Fairouz Tchier

SESSION: SOFTWARE MAINTENANCE & UNDERSTANDING

Managing Fine–grained Changes in Software Document Relationships 681
Tien Nguyen

Defects in Open Source Software Maintenance – Two Case Studies: Apache and Mozilla 688
Virpi, E Hotti, Timo, P Koponen

Eliciting a Model of Emergency Corrective Maintenance at SAS 694
Mira Kajko–Mattsson, Per Winther, Brian Vang, Anne Petersen

SESSION: SOFTWARE PROCESS

Towards A Better Understanding Of Process Patterns 703
Hanh Nhi Tran, Bernard Coulette, Bich Thuy Dong

Goal–Driven Measurement Framework for Software Innovation Processes 710
Subhas Misra, Vinod Kumar, Uma Kumar

A Personal Software Process Tool for Eclipse Environment 717
Xiaohong Yuan, Percy Vega, Huiming Yu, Yaohang Li

QFD Applied to Software Development 724
Lelis Tetsuo Murakami, Edison Spain, Jose–Sidnei Martini

Introducing Personal Software Process in A Small Computer Science Program 731
Xiaohong Wang

SESSION: PROCESS ANALYSIS + SOFTWARE AGENTS

Improving Prediction Accuracies Using Data Imputation 741
Sumanth Yenduri, Sitharama Iyengar, Louise Perkins

Legacy System Reengineering: Essential Process Steps 748
Gregory C. Arnold, Theresa Jefferson

Framework For Separation of Performance Concerns and Improved Modularity in Multi Agent Systems Using Aspects 754
Tariq Mehmood, Naveed Ashraf, Khalid Rasheed

Understanding Scalability Issues for a Distributed Simulation Environment Using Intelligent Coordinated Entities 758
April Crockett, Rodger Maarfi, Srini Ramaswamy, Eric Brown, Mike Rogers

A Fuzzy Agent System that Makes Autonomous Decisions according to State Transitions on Car Driving 764
Chun Kyu Lim, Hyoun Goun Han, Byung Ug Kang

SESSION: REVERSE ENGINEERING

Wire–Tapped Intelligence; Machine Recognition Of Specific Phrases To Nab A Suspect 773
J. S. Mirza, S. A. Hayat

SESSION: IT INTEGRITY

IT Integrity 783
Deborah Kobza

SESSION: SOFTWARE PROJECTS: DEVELOPMENT ASPECTS &EXPERIENCE REPORTS

SAI–BOTS: Scripted Artificial Intelligent Basic Online Tank Simulator 793
William Brandstetter, Michael Dye, Jesse Phillips, Jason Porterfield, Frederick Harris, Brian Westphal

Specification of AutomN: An Automatic Protein Sequence to Protein–Protein Interaction Software System 800
Inna Williams, Nicholas Lyle, Sergiu Dascalu, Brian Beck, Kanwal Brar

DiRT – Dust in Real–time: The Specification Process 807
Marcos Bagmy, Ryan Romero, Brett Sulprizio, Hiroko Uda, Joseph Jaquish, Frederick Harris

Thraxion: Three–Dimensional Action Simulator 814
Justin Gerthoffer, Jon Studebaker, David Colborne, Jeff Stuart, Frederick Harris

Component + Aspect = an Extensible and Adaptable System Software

Paniti Netinant[1,2]

[1]Department of Computer Science
Bangkok University
Bangkok, Thailand
netipan@iit.edu

[2]Concurrent Programming Research Group
Computer Science Department
Illinois Institute of Technology
Chicago, IL, U.S.A.
elrad@iit.edu

ABSTRACT

The benefits associated with separation of concerns are well established. Aspect-Orientation is a methodology that aims at separating components and aspects from the early stages of the software life cycle, and using techniques to combining them together at the implementation phase. Component-based programming systems have shown themselves to be a natural way of constructing extensible software. Well-defined interfaces, encapsulation, late binding and polymorphism provide extensibility, yet despite this synergy, components have not been widely employed at the systems level. This is primarily due to the failure of existing component technologies to provide the protection and performance required of systems software. In this paper we identify the requirements for a component system to support extensions, and describe an extensibility and adaptability in the design of system software. We discuss an aspect-oriented framework that can simplify system design by expressing it at a higher level of abstraction. Our work concentrates on how to achieve a higher separation of aspects, components, and layers from each other. Our goal is to achieve a better design model for system software in terms of extensibility, reuse and adaptability.

Keywords— Adaptability, Aspect-Oriented Framework, Extensibility, System Software.

1. System Software Design Issues

The commercialization of system software, such as operating systems, has resulted in their design gaining more importance. Operating systems are constantly extended for improvements as well as to support new features and hardware. During design of system software, certain decisions such as reusability and adaptability [5] are crucial. Extensibility provides the capability to either change current features or to support new features. Availability is the ability of the system to providing the service over a period of time. Stability is the capability of the system maintaining the correct service in every same state.

System software design issues include distributed algorithms, naming, resource allocation, distributed operating systems, system integration, reliability, tools and languages, real-time systems and performance measurement.

2. Separation of Concerns

The principle of separation of concerns lies at the heart of software development as it introduces a number of benefits, originally addressed by [4, 14]. These include better understanding, extensibility, debugging of the system, and better reuse of the concerns. Although these benefits have been well established, there is still no universally accepted methodology in order to guide a programmer how to best achieve it. The system designer has to consider how a number of aspects in the system can be captured, and how a separation of concerns [14, 9] will be addressed both in the design and implementation of the system software.

Functional decomposition has so far been achieved along one dimension, based on the underlying paradigm. In Object-Oriented Programming (OOP), this dimension is a component hierarchy that includes methods, objects and classes. Current programming languages and techniques have been

portive to functional decomposition. Further, ...ning system design has also been aligned with ...itional functional decomposition techniques. ...wever, no functional decomposition technique ...yet managed to address a complete separation ...concerns. OOP seems to work well only if the ...blem can be described with relatively simple ...rfaces among objects. Unfortunately, this is not ...e case when we move from sequential program...ng to concurrent and distributed programming.

...s system software becomes larger, the interac...n of their components is becoming more com...x. The interaction may limit reuse, comprehen...n, and make the system software difficult to ...luate the design and correctness.

Certain properties of the systems do not localize ...ll. Rather, they tend to cut across groups of ...ctional components, making the system difficult ...understand and evolve. Features such as sched...ng, synchronization, fault tolerance, security, ...d balancing, performance measurement, testing ...d verifications are all expressed in such a way ...t they tend to cut across groups of objects. This ...ngling of concerns [10] results in an increase of ...e dependencies between functional components ...t makes their source code difficult to under...nd, evolve, and maintain. As a result, simple ...ject interfaces are violated and the traditional ...OP benefits no longer hold.

One current attempt to resolve this issue is the ...pect-Oriented Software Architecture. We distin...sh between components and aspects in the de...gn of system software. Aspect-Oriented Pro...amming (AOP) [7, 8, 10] is a methodology that ...ggests a separation of components and aspects ...om the early stages of the software life cycle as...cts should be addressed relatively separately ...om the functional components. At the implemen...tion phase, different AOP technologies use ...echanisms to combine aspects and components ...gether.

In this paper we address a number of system ...sign issues based on an aspectual decomposition ...d in the context of the support provided by As...ect-Oriented Frameworks [1, 2, 13, 14]. Our goals ...re to provide a better design for system software ...mpared with what has so far been able to be sup...rted by traditional approaches, better flexibility ...d higher reusability, as well as to provide a tech...ique that would be practical to implement.

3. An Aspect-Oriented Frameworks

In [11, 12] AOP is viewed as a general framework for separating the concerns in the system software. Dijkstra introduced the layered approach for the design and implementation of operating systems [3]. The layered approach, consisted of layers and components, has showed all the advantages of the modular design. Our approach addresses the separation of components and aspects from the early stages of design, and uses a framework to capture the creation of system aspects as well as the coordination of components and aspects (as well as inter-aspect coordination) during runtime.

Our proposed framework is based on decomposition of aspects in system design that consists of components, aspects, and layers. On top of components and aspects, a layer consists of a collection of components and aspects. In general, lower layers deal with system software. The higher layers deal with application software. The higher layer, the closer the specification and requirement are of the application software.

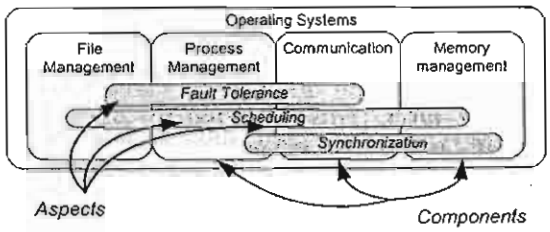


Figure 1. Fault tolerance and scheduling aspects cut across components in the operating systems such as file system, process management, and communication.

By separating the different aspects of each component, we can separate components, aspects, and layers from each other. It would thus be possible to abstract and compose them to produce the overall system. This would result in the clarification of interaction and increased understanding aspects of each component in the system design and implementation. High-level of abstraction is easier to understand. Further, the reusability achieved by the higher level can use the lower level of the implementation not only to promote extensibility and refinement, but also to reduce cost and time in system development. A change in the implementation at a lower level would not result in

change at the higher level if the interface level has not been changed. Thus the design can achieve stability, consistency, and separation of concerns. An aspect might have multiple domains. Some aspect (scheduling, synchronization, naming, and fault tolerance e.g.) is scattered among many components in the system with varying policies, different mechanism, and possibly under different applications. Therefore, an aspect can be redefined to meet the specific requirement.

3.1 Example

The proposed Aspect-Oriented Frameworks for system software [13, 14, 15] is an extended model of the Aspect Moderator Framework [1, 2].

The overall framework architecture is divided into two frameworks based on two layers: a base framework on the lowest layer and an application framework on the upper layers. The Base Framework corresponds to the system layer. On the upper layer(s) we may have more than one application frameworks.

The framework uses design patterns [6]. In this framework, aspects are created using the Abstract Factory and the Bridge patterns. The Abstract Factory would isolate aspects from implementation classes because the factory encapsulates the responsibility and the process of creating aspect objects. The class of concrete aspect appears only once in a functionality, where it's instantiated. The framework promotes consistency when an aspect is modified. The Bridge pattern avoids a permanent binding between an abstraction and its implementation. An example where this would be beneficial is when an implementation concern must be selected, or switched at run-time. This way, different aspect abstractions and implementations can be combined and extended independently. This implementation is still useful when a change in the implementation of a class must not affect its existing components. As a result, a class need not be recompiled, but just re-linked. This approach supports polymorphism, and manages to avoid proliferation. Changing the implementation of an aspect abstraction should have no impact on functionality either. A smart-protection proxy controls access to the aspects and allows additional housekeeping tasks when an aspect is accessed.

In the application framework, the Adapter pattern allows the aspect factory to either convert the interface of an existing aspect (super aspect or aspects in the lower layers) into another interface functionality expect or to create a new aspect. Ideally, a new aspect should reuse an existing aspect to create new aspects, when it could be used. The upper layer can redefine existing aspects and override them.

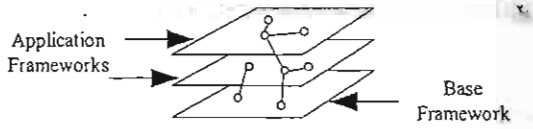


Figure 2. Aspect-Oriented Framework.

3.2 Adaptability of Framework

The general architecture of the framework promotes reusability (the upper layer can reuse aspects from the lower layer), extensibility, and ensures adaptability of aspects and components because both are designed and implemented relatively separately from each other. Aspects in the application framework can be extended and redefined by aspects provided by the layer to meet new requirements. A new aspect can be added in both system layer and application layer without interfering with aspects or components in other layer. The Aspect-Moderator in both frameworks need not be modified when a new aspect is introduced.

4. Discussion

The Aspect-Oriented Design Framework is a three-dimensional model (ACL) that consists of a collection of aspects, components, and layers. Aspects can provide the cross-cutting abstractions. Components form the main functionality of the system. Layers can be divided into lower, intermediate, and upper level. The lower level represents the operating system providing reusable primitives for the intermediate and upper levels. The intermediate level corresponds to the system programming or interface definition. The upper level corresponds to application and programming level.

4.1 Reusability, Stability and Consistency

The abstraction of an aspect in the lower level provides transparency. The upper aspects or components can use the lower aspects or components

ithout knowing the internal details of how the wer aspects or components are implemented. In- mation hiding promotes either component or ect modifiability and simplifies the perception the upper level. The upper level component or plication can use the abstraction of aspect and mponent in the lower level without knowing the ternal details of how the lower level aspects are nstructed. If the lower aspects are changed (to prove performance or to add new features, for ample), provided the aspect interface (intermedi- e level) remains constant, the upper level aspect ed not change. This approach could result in bet- r modifiability of the system.

42 Extensibility

Software reusability not only can save time in ogram development, but it can also avoid unnec- ary proliferation of functions. By reusing of oven and debugged high quality software, it will duce problems after a system becomes opera- nal. Polymorphism enables us to provide a gen- ality of aspect to handle a wide variety of poli- s. It also makes it easy to add new capabilities to aspect or to improve performance of compo- nts. It will help to deal with complexity and re- dancy in the system, and could be particularly fective for implementing layered software sys- ms. When fundamental aspects of the system ch as scheduling, synchronization, and fault tol- nce are created, we can define them as a super- pect. When creating a new aspect (to change or d a policy for example) we can designate that a w aspect is to either inherit from or override its per-aspect rather than being re-written. This new pect is refereed to as a sub-aspect.

43 Adaptability

An aspect abstract is a super-aspect provided by e system and it can be reused and redefined by b-aspects in the upper levels. The sole purpose of abstract aspect is to provide an appropriate su- r-aspect from which other aspects could inherit, erride or redefine implementation. Process man- gement in the operating systems can redefine hedaling by round robin. Communication com- nent might need FCFS. The file system would en need to use the same policy as process man- gement. For example, a database application does a scan of one portion of its memory, while doing random access to another portion. A scheduling of a file system implementation using LRU replacement policy will perform poorly on the scanned memory. A scheduling of a file system should be capable to reconfigure to an appropriate policy for a better performance.

4.4 Architectural and Language Independence

The framework provides an independent architecture and language because each framework is not a modular unit such as a procedures or an object. System design should begin by focusing attention on patterns that can be used to solve the similar problem, postponing considerations of architecture and language constructs. At the implementation level, Aspect-Oriented Framework can be modeled in abstractions like classes in Object-Oriented Programming or aspects in Aspect-Oriented Programming.

5. Conclusion

System software design should not be seen as a two-dimensional model consisting of layers and components that includes single monolithic aspects. In this paper, we stressed the importance of the better separation of concerns within the context of an Aspect-Oriented Framework and we discussed how this technique could provide an alternative to system software such as operating system design and implementation. Our work concentrates on the decomposition of aspects and components in software systems and our goal is to achieve a better design and implementation of system and application software. Our design framework provides an adaptable model that allows for an open language where new aspects can be manageable and added in both application and system software easier. The interaction of newly added aspects is specified by a contract that binds a new aspect to the rest of the system rather than having to re-engineer the whole system. The framework approach is promising, as it seems to be able to address a large number of system and application aspects. The advantage of decomposing of functional components and aspects in every layer is that it promotes comprehension, reusability, adaptability and manageability of both

components and aspects in system and application software easier.

References

[1] Constantinides C. A., A. Bader, T. Elrad. A Framework to Address a Two-Dimensional Separation of Concerns. Position paper to OOPSLA Workshop on Multidimensional Separation of Concerns, np., Denver, CO, November 1999.

[2] Constantinides C. A., A. Bader, T. Elrad, M. E. Fayad, and P. Netinant. Designing an Aspect-Oriented Framework in an Object-Oriented Environment, *ACM Computing Surveys*, Vol. 32, No. 1es, Article No. 41, March 2000.

[3] Dijkstra E. W. The Structure of THE Multiprogramming System. *Communications of ACM*, pp. 341-346, May 1968.

[4] Dijkstra E. W. *A Discipline of Programming*. Englandwood Cliff, NJ: Prentice-Hall, 1976.

[5] Fayad M. E., M. Cline. Aspect of Software Adaptability. *Communications of ACM*, Vol. 39, No. 10, pp. 58-59, 1996.

[6] Gamma E., R. Helm, R. Johnson, and J. Vlissides. *Design Pattern: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[7] Kiczales G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. *ACM Computing Surveys*, Vol. 28, No. 4es, Articles No. 154, np., December 1996.

[8] Kiczales G., J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors. *Proceedings of the 11th European Conference on Object-Oriented Programming, number 1241 in Lecture Notes in Computer Science*, pp. 220-242, Finland, June 9-13 1997. ECCOP'97, Springer Verlag, Berlin.

[9] Lopes C. V. and W. L. Hursch. *Separation of Concerns*. College of Computer Science, Northeastern University, Boston, February 1995.

[10] Lopes C., B. Tekinerdogan, W. de Meuter, and G. Kiczales. *Aspect-Oriented Programming*. In M. Aksit and S.Matsuoka, editors, Proceedings of the 12th European Conference on Object-Oriented Programming ECCOP'98, Springer Verlag, 1998.

[11] Lorenz H., David, Visitor Beans; An Aspect-Oriented Pattern ECOOP Workshop on Aspect-Oriented Programming, np., 1998.

[12] Mens K., L. Cristina, T. Badir, and G. Kiczales. Aspect-Oriented Programming. ECOOP Workshop report on Aspect-Oriented Programming, np., 1997.

[13] Netinant P., and T. Elrad. Improving Concurrent Object Interactions Using Aspect Orientation. *Proceedings of the International Conference on Software Engineering Research and Practice (SERP 2004)*, Las Vegas, Nevada, USA, June 2004.

[14] Parnas D., On the Criteria to be Used in Decomposing Systems into Modules. *Communications of ACM*, Vol. 15, No. 12, pp. 1053-1058, December 1972.

Proceedings of the ISCA
15th International Conference on

SOFTWARE ENGINEERING

AND DATA ENGINEERING

Los Angeles, California, USA
July 6 - 8, 2006

Editors: W. Dosch and W. Perrizo

A Publication of
The International Society for
Computers and Their Applications - ISCA

ISBN: 978-1-880843-59-5

The Proceedings of the ISCA 15th International Conference on Software Engineering and Data Engineering (SEDE-2006), held in Los Angeles, California, USA, July 6-8, 2006.

SPONSOR: The International Society for Computers and Their Applications – ISCA

General Chair: Narayan Debnath, Winona State University, USA

Program Chairs:

Software Engineering: Walter Dosch, University of Lübeck, Germany
Data Engineering: William Perrizo, North Dakota State University, USA

Program Co-Chairs:

Software Engineering: Annette Stümpel, University of Lübeck, Germany
Data Engineering: Imad Rahal, College of St Benedict | St. John's University, USA

INTERNATIONAL PROGRAM COMMITTEE

Hamid R. Abachi, Monash U., Australia
Isham Al-Mubaid, U. of Houston, USA
Noureddine Belkhatir, U. of Grenoble, France
Mark Burgin, UCLA, USA
Wang Canton, Skipanon Inc., USA
Maurice Cohen, U. of California, USA
Alfredo Cuzzocrea, U. of Calabria, Italy
Sergiu Dascalu, U. of Nevada-Reno, USA
Anne Denton, North Dakota State U., USA
Qiang Ding, Jiangsu Telecomm. Inc., China
Qin Ding, Penn. St. U, Harrisburg , USA
Wenying Feng, Trent U., Canada
Jocelene Fouchal, U. of Antilles-Guyane, France
Hossein Hakimzadeh, U. of Indiana, S Bend, USA
George Hamer, South Dakota State U., USA
Ramzi Haraty, Lebanese American U., Lebanon
Gongzhu Hu, Central Michigan U., USA
Marina Hudson, U. of California, USA
William Jockheck, Northern State U., USA
Tatjana Kapus, U. of Maribor, Slovenia
Martin Kempa, sd&m, Germany
Dieter Kranzmüller, U. of Linz, Austria
Stephen Krebsbach, Dakota State U., USA
Gordon Lee, San Diego State U., USA
Sönke Magnussen, Lufthansa Revenue Services, Germany
Eda Marchetti, U. of Pisa, Italy
Merik Meriste, Tartu U., Estonia
Pascale Minet, INRIA, France
Pornsiri Muenchaisri, Chulalongkorn U., Thailand
Mara Nikolaidou, U. of Athens, Greece
M. Mehdi Owrang O., American U., USA
Fei Pan, U. of Southern California, USA
Jaan Penjam, Tallinn Technical U., Estonia
David Pheanis, Arizona State U., USA
Dongmei Ren, IBM Inc., USA
Daniel Riesco, U. Nacional de San Luis, Argentina
Kirk Scott, U. of Alaska, USA
Howard Sholl, U. of Connecticut, USA
Annette Stümpel, U. of Lübeck, Germany
Elizabeth Wang, Waynesburg College, USA
Lok Yeung, Lingnan U., Hong Kong
Qinghua Zou, Microsoft Research, USA
Zhili Zhang, Tibco, USA

ADDITIONAL REVIEWERS:

Chermsak Buntha, U. of Nevada, Reno, USA
Harry Gros-Désormeaux, U. des Antilles et de Guyane, France
Mike McMahon, Jr., U. of Nevada, Reno, USA
German Montejano, U. Nacional de San Luis, Argentina
Ando Saabas, Tallinn U. of Technology, Estonia
Abbas Tarhini,U. des Antilles et de Guyane, France

All published papers have been peer reviewed.
This publication is abstracted and indexed in INSPEC and DBLP.

ISCA, 975 Walnut Street, Suite 132, Cary, NC 27511 · Ph: (919) 467-5559 · Fax: (919) 467-3430
E-mail: isca@ipass.net WWW site: http://www.isca-hq.org

Preface

The 15th International Conference on Software Engineering and Data Engineering (SEDE-2006) provides an international forum for scientists throughout the world to present research results in the areas of software engineering and data engineering. The conference particularly welcomes contributions at the junction between theory and practice with immediate impact on applications.

This year's SEDE conference is located in Los Angeles, the second largest city in the USA originally founded under the name "El Pueblo de Nuestra Senora la Reina de los Angeles". Today Los Angeles forms a large metropolitan area offering attractive beaches along the Pacific shore and Hollywood, the historic home of the movie studios.

SEDE-2006 features two invited talks by prominent scientists and 53 contributed papers including a special session. The conference covers a broad range of topics including theory, methods, applications and tools. The conference runs – apart from the plenary sessions - in two parallel tracks. The software engineering track includes sessions about software project management, re-engineering and software analysis, software quality, software architectures, aspect oriented design, software metrics, verification and real time systems. The data engineering track comprises sessions about data mining, XML and applications, data base design and data warehouses. Two special sessions on applications in software engineering complement the conference program.

Scientists from more than 20 countries submitted 77 papers to the conference. Each contribution was evaluated by at least two, mostly three members of the international program committee and additional referees judging the originality, significance, technical contents, application contents and presentation style. We used the START conference system installed at the Lübeck site to automate the workflow for submission and refereeing.

We gratefully acknowledge the professional work of the international program committee and the sub-reviewers contributing 225 referee reports. We appreciate the dedication of the invited speakers Prof. Michael A. Arbib, University of Southern California at Los Angeles, and Prof. Wesley W. Chu, University of California at Los Angeles, for their contributions. We owe great thanks to Mary Ann Sullivan for the well-organized conference management.

We also want to thank all presenters and attendees for actively contributing to the success of SEDE-2006. We are looking forward to excellent presentations and interesting discussions, which will broaden our professional horizons. All participants are invited to make new friends within the ISCA family.

Welcome to Los Angeles - welcome to SEDE-2006!

Narayan Debnath
Winona State University, USA
Conference Chair

Walter Dosch
University of Lübeck, Germany
SE Program Chair

William Perrizo
North Dakota State University, USA
DE Program Chair

Annette Stümpel
University of Lübeck, Germany
SE Program Co-chair

Imad Rahal
College of St. Benedict / St. John's University
DE Program Co-chair

INTERNATIONAL SOCIETY FOR COMPUTERS AND THEIR APPLICATIONS

15th International Conference on Software Engineering and Data Engineering (SEDE-2006)

July 6 - 8, 2006
Omni Los Angeles Hotel at California Plaza
Los Angeles, California USA

TECHNICAL PAPER INDEX

SOFTWARE ENGINEERING

Traceability for Managing Evolutionary Change
Patrick Maeder, Matthias Riebisch and Ilka Philippow (Technical University of Ilmenau, Germany) 1

Mercury: A Process Management System based on the Agent Technology
Seung Yong Choi and Hee Yong Youn (Sungkyunkwan University, Korea) and Jeong Ah Kim (Kwandong University, Korea) 9

Management Support of Interorganizational Cooperative Software Development Processes based on Dynamic Process Views
Markus Heller and René Wörzberger (RWTH Aachen University, Germany) 15

Agile Plan Refactoring
David Serr and Stephen Clyde (Utah State University, USA) 22

Extending Reverse Inheritance
David Serr (Utah State University, USA) 29

A Strategy to Integrate Legacy Systems
Alfredo Espinosa Reza, José Alfredo Sánchez López, José María Suárez Jurado, Agustín Quintero Reyes (Instituto de Investigaciones Eléctricas, Mexico) 35

A Prototype Decompiler for 32-bit x86 Executables
Hao Liu and Feodor Vainstein (Georgia Institute of Technology, USA) 41

Debugging with Software Visualization and Contract Discovery
S. Kanat Bolazar and James W. Fawcett (Syracuse University, USA) 47

A Method to Improve Software Testability
Yuanping Li, Jianmin Wang and Liang Zhao (Tsinghua University, China) 51

Alternative Approach to Utilize Software Defect Reports
Rattikorn Hewett and Aniruddha Kulkarni (Texas Tech University, USA) 57

Estimating Software Reliability with Static Analysis Techniques
Walter W. Schilling, Jr. and Mansoor Alam (The University of Toledo, USA) 63

Software Defect Fractal Description
Kai Zhang (Zhongnan University of Economics and Law, China) 69

Operational and Program Schemas
M. Burgin (University of California, Los Angeles, USA) 74

ASMADE: Automated Schema MApping for Documents Exchange
Aïcha-Nabila Benharkat (LIRIS-INSA de Lyon, France), Rami Rifaieh (University of California, San Diego, USA), Youssef Amghar, Herzi Khaled (LIRIS-INSA de Lyon, France) 79

A Detailed UML Design of a Software Testing Tool
Narayan C. Debnath, Jesse R. Haakenson (Winona State University, USA) Mark Burgin (University of California, Los Angeles, USA) and Joyati Debnath (Winona State University, USA) 86

A Framework for Requirements Elicitation Techniques Selection
Yirsaw Ayalew (University of Botswana, Botswana) 92

Using UML in a Non-Software Design Task: Creating an Electronic Software Engineering Handbook
Sergiu Dascalu (University of Nevada, Reno, USA), Marcel Karam (American University in Beirut, Lebanon), Muhanna Muhanna and Salyer Reed (University of Nevada, Reno, USA) 98

The Effects of Requirements and Task Uncertainty on Software Product Quality
Ayad Aldaijy (Royal Saudi Air Force, Saudi Arabia) and Khalid A. Buragga (King Faisal University, Saudi Arabia) 104

Correctness as a Relative Gradual Software Property
M. Burgin (University of California, Los Angeles, USA) and N. Debnath (Winona State University, USA) ... 112

Remote Sensing and Prompting for Early Stage Dementia Patients
Donna L Hudson, Maurice E. Cohen (University of California, San Francisco, USA) 116

A High Population, Fault Tolerant Parallel Raytracer
James Skorupski, Ben Weber, and Mei-Ling L. Liu (Cal Poly State University, USA) 122

A Context-Aware Architecture for Railway System
Chia Hung Kao, Hewijin Christine Jiau and Ku Chen Wu (National Cheng Kung University, Taiwan, ROC) 128

JDOSecure: A Security Architecture for the Java Data Objects-Specification
Matthias Merz (University of Mannheim, Germany) 134

Model Checking for Synchronous Java
Duc-Duy VO and Claude Petitpierre (Swiss Federal Institute of Technology at Lausanne, Switzerland) ... 141

A Formal Approach to Requirement Verification
Divya K. Nair, Stéphane S. Somé (University of Ottawa, Canada) 148

Application of AI Planning Technique in Software Engineering
Sung Kim (North Dakota State University, USA) 154

Establishing a Common Modeling Framework using UML to Effectively Support Faster-Than-Real-Time Simulation
Mara Nikolaidou, Vassilis Dalakas (Harokopio University of Athens, Greece) Dimosthenis Anagnostopoulos and George-Dimitrios Kapos (University of Athens, Greece) 158

Design Patterns for Real-time Distributed System
Yiqin Xu, Daisy F. Sang (Cal Poly Pomona, USA) and Chang-Shyh Peng (California Lutheran University, USA) 164

Extending the Rapide ADL to Specify Aspect Oriented Software Architectures
Karen Palma , Yadran Eterovic (Pontificia Universidad Católica de Chile, Chile) and Juan Manuel Murillo (Universidad de Extremadura, Spain) 170

Extensibility Aspect-Oriented Framework to Build Agent-Based System Software
Paniti Netinant (Bangkok University, Thailand and Illinois Institute of Technology, USA) 177

An Empirical Research of the Software Project Measures Model
Yeonshick Ahn (Kyungwon College, Korea) 183

Identification of Suitable Metrics for Reuse Oriented Software Products using the Methods of Attribute Relevance Analysis
Jasmine K.S (R. V. College of Engineering, India) 189

DATA ENGINEERING

Biological, Intelligent Text-Based Ranking of Genes
Imad Rahal (College of St. Benedict | St. John's University, USA), Walid Saeed, Arun Srivastava, Pratap Kotala, Ranapratap Syamala, William Perrizo, and Cesar Carvalho (North Dakota State University, USA) 193

Fault Tolerant Control Using a Generalized ANFIS Structure and Evolutionary Tuning
In Soo Lee (Sangju National University, Korea) and Gordon K. Lee (San Diego State University, USA) 199

BioFacets: Integrating Biological Databases using Facetted Classification
M. Mahoui, Z. B. Miled, A. Godse, H. Kulkarni, and N. Li (IUPUI, USA) 205

A Hierarchical Approach for Clusters in Different Densities
Baoying Wang (Waynesburg College, USA) and William Perrizo (North Dakota State University, USA) 211

Integrating Statically Typechecked XML Data Technologies into Pure Java
Henrike Schuhart, Beda C. Hammerschmidt, and Volker Linnemann (University of Lübeck, Germany) 217

Translating XSLT into XQuery
Albin Laga, Praveen Madiraju, Darrel A. Mazzari and Gowri Dara (Marquette University, USA) 223

XINDEX - XPATH Indexing Specification for XML
Kevin Ricords and Qin Ding (Pennsylvania State University - Harrisburg, USA) 228

Representation of Accounting Standards : Creating an Ontology for Financial Reporting
Pierre Teller (University of Nice, France) 234

Real Time Self-Maintenable Update to Aggregate Information for Data Warehouse
Clemente García (Instituto Tecnológico de Culiacán, Mexico) and Matilde Celma (Universidad Politécnica de Valencia, Spain) 240

SlidingCubes- Mining for Bigger Dense Regions in Sparse Data Cubes
Shahzad Majeed Tiwana (University of Southern California, USA) 246

A Predicate-based Incremental Refresh Method for a Data Warehouse
Dongmei Ren, Guogen Zhang (IBM Silicon Valley Lab, USA) and William Perrizo (North Dakota State University, USA) 253

Query Optimization for Distributed Data Streams
Ying Liu and Beth Plale (Indiana University, USA) 259

The P-list for Orthogonal Range Search
Bradford G. Nickerson and Qingxiu Shi (New Brunswick University, Canada) 265

Design Patterns Across Software Engineering and Relational Databases
Cyril S. Ku (William Paterson University, USA), Thomas J. Marlowe (Seton Hall University, USA) and Nathan M. Mantell (William Paterson University, USA) 271

DNA Sequence Encoding for Relational Storage and SQL Query
Qinghua Zou (Microsoft Corporation, USA) and Raymond K. Pon (University of California, Los Angeles, USA) 275

Automating Technical Indicators in the Financial Market
Harshpreet S. Walia and James W. Hearne (Western Washington University, USA) 281

A Projected Clustering Algorithm in High Dimensional Space
Ping Deng, Weili Wu, Yaochun Huang, Zhongnan Zhang (The University of Texas at Dallas, USA) 286

Chi-Squared Statistical Steganalysis of Database Tables
George Hamer (South Dakota State University, USA) and William Perrizo (North Dakota State University, USA) 292

SkiPeR: A Family of Distributed Range Addressing Spaces for Peer-to-Peer Systems
Antonios Daskos, Shahram Ghandeharizadeh, Ramin Shahriari (University of Southern California, USA) 298

RFID Systems: An Overview and Open Research Issues
C. Parikh, A. Zeid and S. Kamarthi (Northeastern University, USA) 304

Time-based Workflow Mining
Deniz Canturk (STM Savunma Teknolojileri Mühendislik ve Tic. A.S., Turkey) and Nihan Kesim Cicekli (Middle East Technical University, Turkey) 310

Extensibility Aspect-Oriented Framework to Build Agent-Based System Software*

Paniti Netinant[1,2]

[1]Computer Science Department
Bangkok University
Bangkok, Thailand
paniti.n@bu.ac.th

[2]Computer Science Department
Illinois Institute of Technology
Chicago, IL, USA.

Abstract

Concurrent real-time software systems are vulnerable to performance saturation and reliability concerns due to environmental influences. Building intelligent concurrent systems that are able to adapt to environmental changes and reconfigure themselves is the key to avoiding performance degradation of concurrent real-time software systems and ensuring the levelness property of such systems. In this paper we present a machine learning-based approach that addresses the design of agent-based intelligent concurrent software systems in order to ensure the reliability and performance properties for such systems. Although reliability and performance are conflicting requirements in most cases, we will show how to use an aspect-oriented technology by which these requirements can be designed, implemented, reused, and replaced in isolation from each other. The performance and reliability of the software system can be reasoned about by intelligent agents who can direct the system to reconfigure itself in order to adapt to the environment changes. The agents rely on the data-mining techniques to discover patterns of performance degradation or imminent signals of reliability violation and to predict policies that cope best with the environmental fluctuations.

Keywords: Framework, Agents, Reusability.

1. Introduction

Concurrent real-time systems are designed mainly in order to ensure performance and the stringent reliability requirements, sometimes referred to as quality properties. Until recently, the design and development of these systems have inter-mixed the quality code with the functionality code for such systems; the quality properties cut across the functional components. This crosscutting phenomenon [1, 2, 5 and 6] generally breaks the component model, and makes it hard to design, and reuse, especially when the functional and non-functional requirements change. Generally requirement changes force reengineering for these systems. Generally, the functional components for these systems are stable. On the other hand, the quality requirements are volatile and reactive to the environment changes. Another issue that necessitates engineering reconfigurability when designing these systems is the make It possible to build these systems so that and real-time properties can be replaced while the systems are running, in order to adapt to environmental changes.

Building intelligent software systems that have open architectures, which support reconfigurability, is essential for concurrent real-time applications by which their well-being and performance are heavily dependent on their capability to cope with the environment fluctuations. The Mars pathfinder problem [9 and 11] is a classical example of such systems where conflict of interest between performance and liveness properties has forced system reset that was due to a priority inversion problem. The Mars Pathfinder spacecraft's engineers were aware of the priority inheritance solution for such problems but they preferred not to use, since it may cause performance degradation for the spacecraft. The lack of software adaptability hooks is the main reason that NASA system engineers chose not to deploy priority inheritance mechanism in order to avoid the system resetting.

Aspect-Oriented Programming (AOP) [4 and 6] is a new programming paradigm that attempts to separate the functional components from the interaction components (aspects). Aspects are defined as properties that cut across groups of functional components. While these aspects can be thought about and analyzed relatively separately from the basic functionality, at the implementation level they must be combined together. Programming concerns manually into the system's functionality using current component-oriented languages results in aspects being tangle throughout the code. This code tangling makes the source code difficult to develop, understand and evolve because it destroys modularity and reduces software quality [8]. In this paper we show how to deploy aspect-oriented technology, which provides an architectural support for the design and development of intelligent concurrent systems. We show how the aspect code can be isolated from the functional components that otherwise would be

* [1] This research has been supported by Thailand Research Funding (TRF) organization and Bangkok University. The contract is MRG4780168.

intermingled with the code of the functional components. Isolating the functional components from the non-functional components, the aspect code, has many attractive benefits: first and foremost it promotes reusability for the functional classes and the aspect classes. It also simplifies the design of complex systems, since the interaction code is separated from the functional code. Our approach is a step toward building reconfigurable intelligent systems and improves the software quality as it complements the object-oriented and component-oriented technologies with a set of design principles in order to engineer adaptability into software systems.

In our approach, assembling the intelligent concurrent systems from both functional and non-functional components through the use of aspect-oriented technology supports both static and dynamic adaptability when building the intelligent concurrent systems.

Recent advances in information technology have demonstrated that there are numerous issues that may affect the quality of service for software systems and that the only way to improve the quality of these services is by using software agents that monitor, advise, and react based on the environment changes. A software agent is an autonomous software component that can react to and interact with its environment. An agent is autonomous, since it runs in its own thread of control, and reactive, because of its capability to respond to incoming messages.

The agent-based approach is still in its infancy, although it is getting more popular as more IT applications are using this approach. Enterprise applications, B2B applications, Personal Agents, Information filtering, Information monitoring, and Interface agents / personal assistants just to name a few. For example, in information monitoring, activities are dependent on the timely notification of changes in the environment or in the data sources. Agents are very useful for monitoring different data sources for specific data. Agents can be dispatched to remote locations to monitor data sources. An interface agent is a program that is able to operate within a user interface and actively assist the user in operating the interface and manipulating the underlying system. MS-Office assistants are an example of this category.

Our experience shows that agent-based approaches are a key component in building an intelligent concurrent system. They complement the aspect-oriented and object-oriented technologies, while an aspect-oriented solution complements object-oriented technology to solve the code-tangling phenomenon and improve code reusability. The agent-based approach complements these technologies in order to support dynamic adaptability and ensure quality of services. Frameworks capture design decisions that are common to applications in certain domains. Generally, frameworks emphasize design reuse over code reuse, although a framework may have concrete subclasses that can be used immediately. In this paper we present a framework that can be used to build intelligent concurrent systems. The key contribution of this work is to show how to deploy aspect-orientation in the design of these systems so that system requirements that may have an impact on performance, reliability, and security are isolated from the functional components, and intelligent agents are deployed to watch each one of these aspects. As we will be described in the subsequent sections, aspect-orientation within our framework helps to engineer reconfigurability into the intelligent systems such that policies can be altered, reused, or replaced without halting the running system. We will also show how to design the intelligent agents based on data mining techniques in order to audit and guide the real-time system from its own training data and to update its knowledge base in real-time.

2. Agents for Aspect-Oriented Concurrent Systems

Concurrent object-oriented open software systems are composed of functional requirements and concurrency requirements. Mixing the functional code and concurrency may impede code reuse; this has been documented in the literature as the inheritance anomaly problem. Solutions for the inheritance anomaly problem vary from domain specific languages (ABCL) to framework based solutions [6 and 7]. Framework solutions are preferred over domain specific languages, since these frameworks are based on common object-oriented languages and require less time to learn and deploy. Despite the success of the framework based approaches, support of static and dynamic adaptability for concurrent software systems has not been addressed in a formal way.

Recent aspect-oriented approaches [4, 5, 6, and 7] have provided an elegant solution to support the static adaptability aspect for concurrent software systems, though the dynamic adaptability aspect has been left un-addressed. Aspect-oriented technology complements the object-oriented technology in order to avoid so-called code-tangling phenomena and support static adaptability for concurrent open software systems; Adaptability [3] is an important factor that enables software systems to evolve in order to meet future requirements. Reflective approaches [8 and 9] offer solutions to support the dynamic adaptability aspect for open software systems, though reflection-based solutions have hard-coded decision processes that react and adapt to environment changes in an intelligent way.

Agent-oriented technology is getting more popular as more industrial applications start to deploy this approach. Agents are needed mainly to deal with uncertainty and react to environment changes and they are very useful to monitor systems resources and notify

the interested parties to change their behavior to cope with the environment changes. Methods to engineer intelligence and machine learning within agents vary from data mining techniques to q-learning [10 and 11]. In [11], the authors presented an approach based on the data-mining technique to discover patterns of behavior and network intrusion detection. And in [12] the authors have demonstrated how q-routing algorithms can be used to discover best routes in a highly congested network.

Our approach integrates agent-oriented technology and aspect-oriented technology to build intelligent concurrent software systems [6]; systems that run within an uncertain environment and require the capability of altering their components and policies during run time. Figure 1 shows our integrated view of these technologies. Our approach delivers a framework solution for building these systems. The agents are needed to assist in the decision making process for reconfiguring the software system during run time. Aspect-orientation techniques are used since they isolate the functional components from the aspectual components like performance and reliability.

Figure 1. A concurrent object as a cluster of components and aspects within the aspect moderator framework.

The aspect-oriented approach [4] has demonstrated its effectiveness in building concurrent object-oriented systems, where the concurrency aspects, like synchronization constraints and scheduling policies, are isolated from the functional components. This approach helps in building a stable software system that can easily adapt to meet future requirements and react to environment changes.

The approach stated in [4] did resolve the static adaptability problem, but did not address the dynamic adaptability aspect for building intelligent concurrent software systems. Our research has revealed the need to have intelligent components, agents that can aid the concurrent software system in the decision-making process to reconfigure itself in order to adapt to environmental changes. The agents in our approach deploy the data mining technique and the Bayesian algorithm to monitor system resources and predicate patterns of performance degradation or imminent signals of reliability violation, and offer advice to react to these changes. Dealing with environmental uncertainty is the key challenge for concurrent real-time systems. Environmental uncertainty may have an impact on performance, reliability, throughput, or quality of service guarantees. One way to monitor system resources and environment concept drifts is by gathering data about usage of systems resources. In our framework we apply Bayesian algorithms and online learning techniques to predict environment changes and adapt to these changes. For example, when a buffer is more than half-full, and the ratio of the number of waiting puts to the number of waiting gets is 1 to 10, we may still prefer put over get, if though the immediate preference shall be given to get, the historical data demonstrates that whenever the buffer is more than half full, the number of waiting gets was substantially greater than waiting puts. This distinction between immediate preferences and desires and long-term well-being objectives has been fully discussed and advocated in the artificial intelligence discipline. Our research has revealed that such distinction is extremely important in building open software systems that operate in volatile environments, where system resources can go through over utilized and underutilized cycles.

Through the support of agents, our framework can be used to build concurrent open software systems that can dynamically adapt to environmental changes and deal with uncertainty. Agents are used to monitor certain aspects of the software systems; scheduling is an example of such aspects. Agents have a knowledge base that they consult to predict the system behavior, and update their knowledge base to keep pace with current environment settings

3. Architecture Of The Framework

Our observation suggests that an Aspect-Oriented Systems (AOS) that uses Aspect-Oriented Framework could support designers and programmers in cleanly separating components and system aspectual properties from each other. Our framework is based on Aspect-Oriented techniques and layered approach [1]. We argue that system aspectual properties of the operating system should be excluded from the system components or services if there is a possibility to often change it, and it should not be treated as a single monolithic aspect. Our proposed framework (CAL) is based on system aspectual decomposition of crosscutting concerns in operating system design and implementation. CAL framework consists of two frameworks: Based Layer and Application Layer Framework.

The aspect-oriented framework supports both vertical and horizontal compositions. Functional and aspectual property components in the framework can be composed

vertically or horizontally. In vertical composition, the upper layer can use the lower functional or aspectual property components from the lower layer. In horizontal composition, functional and aspectual property components in the particular layer only use to be composed.

The framework is based on system aspectual decomposition of crosscutting concerns in operating system design and implementation. The framework consists of two frameworks: The Based Layer and The Application Layer Framework. A system aspectual property is implemented in the SystemAspect class, while a component of the system is implemented as a Component class. Alike AspectJ [11], our framework uses *PointCut, Precondition,* and *Advice*. The framework uses *PointCut, Precondition,* and *Advice*. The AspectModerator class, where the point cut is defined, combines both system aspectual properties and components together at runtime. Pointcuts are defined collections of join points, where system aspectual properties will be altered and executed in the program flow. Every aspectual property can identify and implement preconditions. A precondition is defined a set of conditions or requirements that must hold in order that an aspect may be executed. Advice is a defined collection of methods for each aspectual property that should be executed at join points. Advice can be either *before* or *after* advice. Before advice can be implemented as *blocking* or *non-blocking*. Before advice is executed when the join point is reached, before the component is executed, if the precondition holds. After advice is executed after the component at the join point is executed. Every aspectual property will define advice methods. Figure 2 and 3 illustrated the execution model of a pointcut in the framework based on inter-dependency and intra-dependency.

In this paper, we show how producers/consumers problem can be implemented in the based layer framework. A system aspectual property is implemented in SystemAspect class, while a component of the system is implemented as Component class. Alike AspectJ [9], our framework uses PointCut, Precondition, and Advice. AspectModerator object, where the point cut is defined, combines both system aspectual properties , and components together at run-time.

Pointcut is defined collections of join points, where system aspectual properties will be altered and executed in the program flow. Every aspectual property could identify and implement precondition. Precondition is defined a set of conditions or requirements that must be hold in order to be executed an aspect. Advice is defined collections of methods for each aspectual property that should be executed at join points. Advice could be either before or after. Before advice could be implemented as blocking or non-blocking. Before advice executes when join point is reached, before the component executed, and if the precondition is hold. After advice executes

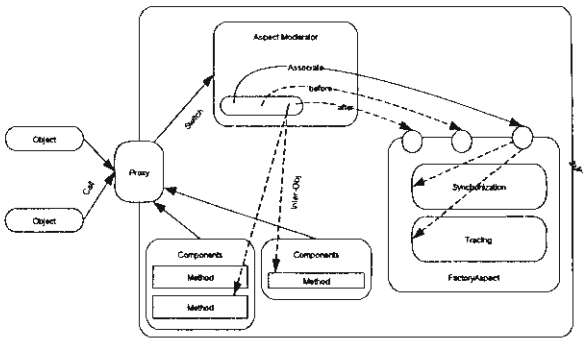


Figure 2. PointCut Defines Inter-dependency

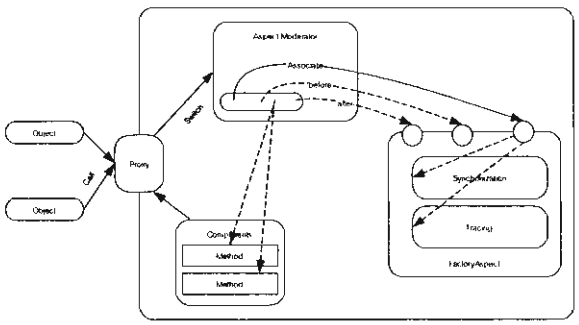


Figure 3. PointCut Defines Intra-dependency

after the component at the join point executes. Every aspectual property will define advice methods. Figure 3 and 4 are illustrated the execution model of a pointcut in the CAL framework based on inter-dependency and intra-dependency.

Our proposed framework (CAL) is based on system aspectual decomposition of crosscutting concerns in operating system design and implementation. CAL framework consists of two frameworks: Based Layer and Application Layer Framework. In this paper, we show how producers/consumers problem can be implemented in the based layer framework. A system aspectual property is implemented in SystemAspect class, while a component of the system is implemented as Component class. AspectModerator object, where the point cut is defined, combines both system aspectual properties and components together at run-time. A Pointcut is defined collections of join points, where system aspectual properties will be altered and executed in the program flow. Every aspectual property could identify and implement precondition. Precondition is defined a set of conditions or requirements that must be hold in order to be executed an aspect. Advice is defined collections of methods for each aspectual property that should be executed at join points. Advice could be either before or after. Before advice could be implemented as blocking or non-blocking. Before advice executes when join point is reached, before the component executed, and if the

precondition is hold. After advice executes after the component at the join point executes

4. Implementing The Framework

The framework consists of four components comprising the architecture of the framework.

- Each functional object (component) provides its services (methods) stripped of any aspectual properties (for example, no synchronization is included in Buffer objects).
- A proxy object intercepts called methods and transfers the calls to the AspectModerator.
- An AspectModerator object consists of the rules and strategies needed to bind aspects at runtime. Aspects are selected from the AspectBank. The AspectModerator orders the execution of aspects. The order of execution can be static or dynamic. Then, each precondition will be checked whether it is satisfied or not.
- An AspectBank object consists of aspect objects that implement different policies of a variety of aspects.

This section presents the design and development of aspect-oriented framework. The model is presented to demonstrate horizontal composition of the framework. The system service must be implemented as a Component class. The system aspectual property (SystemAspect class) must be derived from the SystemAbstractAspect interface to implement the required behavior of a system aspectual property. A SystemAspectFactory consists of many system aspectual properties such as synchronization, tracing, logging, and reliability. The SystemAspectFactory, derived from the SystemAbstractAspectFactory interface, is known as an aspect bank.

During runtime, each SystemAspectFactory will be associated with one SystemAspect. The AspectModerator class must be derived from the AspectModerator interface to implement the required behavior. The following points are important about the aspect-oriented framework:

- A base layer framework is an implementation of an underlying system.
- An application layer framework is- an implementation of application software over the system software represented by a base layer framework.
- A client object requests a service through a ProxyObject object of a framework.
- A functional component is implemented as a Component class without any aspectual property.
- A SystemAspectFactory object consists of various SystemAspect objects. A SystemAspect object is controlled by a SystemAspectFactory object.
- Each system aspectual property must be implemented as a SystemAspect object.
- Each crosscutting between Component object and a SystemAspect object must be defined in AspectModerator object as joinpoints in a Pointcut method.
- A client requests a service by sending a message to a ProxyObject object. The ProxyObject object changes the request to a specific pointcut method, and forwards it to the AspectModerator object.

The Proxy class is responsible for intercepting and forwarding the message sent from Client object to request a service. The Proxy class must implement the behavior of intercepting a service request. A client object of an aspect-oriented framework must request a service by calling the call() method. A call() method consists of at least two parameters: object name provided a service and a service requested to serve. The first parameter is of type string, and the second is type of string as well. The ProxyObject class will forward a request to the AspectModerator object by calling a PointCut() method. A PointCut() method must have the same number parameters and the same parameter type as the call() method.

The SystemAspectFactor class must be derived from the SystemAspectFactoryAbstract interface to implement the required behavior. The AspectModerator class is responsible for composing the functional components and the system aspectual property into a service request. The AspectModerator class acts like a coordinator between functional components and system aspectual properties, when and where system aspectual properties will be composed into a functional component. The composition of system aspectual properties and functional components must be guided and defined as PointCut() method. Each PointCut() method must have at least two parameters: component name and service name (methods of the component) that will be composed. The first parameter is of type string, and the second is type of string as well.

The AspectModerator class will create the SystemAspectFactory object. The SystemAspectFactory object can support either static or dynamic aspects at runtime. The attachImple() method is used to associate a system aspectual property of a SystemAspectFactory object. The AspectModerator class will be associated with functional components and system aspectual properties that will be composed. The PointCut() method will define join points between functional components and system aspectual property. Currently, the PointCut() method uses if...then...else... statements to define joinpoints. The synchronization aspect property crosscuts both read and write services. It crosscuts the before and after execution of read and write services. A tracing property crosscuts both read and write services. It only crosscuts the after execution of read and write services.

The SystemAspectFactor class must be derived from the SystemAspectFactoryAbstract interface to implement

the required behavior. The SystemAspectFactory class provides a dynamic binding of variety system aspectual properties. It focuses on the interface of the system aspectual property. Each system aspectual property must be derived from the SystemAspectAbstract interface to implement the required behavior. Implementation of a system aspectual property is implemented in the SystemAspect class. Each system aspectual property can define before(), after(), and precondition() methods depending on its needs.

The AspectModerator class operates composition between system aspectual properties and functional components using a composition rule defined by join points of a pointcut. The AspectModerator class performs composition rules by sending AspectFactory messages. Messages sending causes polymorphism. The implementation of AspectFactory uses bridge patterns. A message finds the correct member object of the AspectFactory, and invokes that object. With polymorphism calls, AspectModerator requires less information about each SystemAspect, so the AspectModerator only needs to have the right SystemAspect interface.

5. Conclusion

Agent-based software systems are the next wave in the software engineering discipline. Recent advances in software technology have stressed the need to build intelligent open software systems in order to build dynamically reconfigurable software systems that can deal with environment uncertainty and predict usage patterns for system resources. A key factor for building highly reconfigurable software systems is to identify and isolate the aspectual code from the functional code in order to maximize reusability and facilitate reconfigurability. Aspect-oriented programming is an emerging programming technique that makes it possible to engineer the reconfigurability of software systems. We use software agents in our framework in order to support the decision-making processes for such systems. Our framework based approach integrates aspect-oriented technology and agent-orientation in order to support the dynamic adaptability aspect for intelligent concurrent software systems, where agents can be asked to monitor system properties and react to undesirable events by reconfiguring the software without halting the running system. The Mars Pathfinder resetting problem was due to hard-coded decisions. The Pathfinder software system was not able to reason about the environment and itself. We can build intelligent software systems based on agents and aspect-orientation, such that the decision making process that copes with the environment fluctuations is isolated from the core functional components. Agents can be used to alter the bias of the system, whenever conflicts arise; between performance and reliability.

6. Reference

[1] Dijkstra, Edsger W. The Structure of THE Multiprogramming System. Communications of ACM, Vol. 26, No. 1, pp.49-52, January 1983.

[2] Dijkstra, Edsger W. A Discipline of Programming. Englandwood Cliff, NJ: Prentice-Hall, 1976.

[3] Fayad, M. E., M. Cline. Aspect of Software Adaptabil-ity. Communications of ACM, Vol. 39, No. 10, pp.58-59, 1996.

[4] Kiczales G., J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors. Proceedings of the 11th European Conference on Object-Oriented Programming, number 1241 in Lecture Notes in Computer Science, pp.220-242, Finland, June 9-13 1997. ECCOP'97, Springer Verlag, Berlin.

[5] Kubat, M. (1998), "*A Machine Learning-Based Approach to Load Balancing in Computer Networks*," Cybermetics and Systems Journal, 23, 1992, pp. 389-400.

[6] Lopes C., B. Tekinerdogan, W. de Meuter, and G. Kic-zales. Aspect-Oriented Programming. In M. Aksit and S.Matsuoka, editors, Proceedings of the 12th European Conference on Object-Oriented Programming EC-COP'98, Springer Verlag, 1998.

[7] Netinant P., C. A. Constantinides, T. Elrad, M. E. Fayad. Supporting Aspectual Decomposition in the Design of Adaptable Operating Systems Using Aspect-Oriented Frameworks. Proceedings of 3rd Workshop on Object-Orientation and Operating Systems ECOOP-OOOWS 2000, pp.36-46, Sophia Antipolis, France, June 2000.

[8] Netinant P., C. A. Constantinides, T. Elrad, and M. E. Fayad, Supporting the Design of Adaptable Operating Systems Using Aspect-Oriented Frameworks. Proceedings of the International Conference of Parallel and Distributed Processing Techniques and Applications (PDPTA), pp.271-278, Las Vegas, NV, June 2000.

[9] Parnas, D., On the Criteria to be Used in Decomposing Systems into Modules. Communications of ACM, Vol. 15, No. 12, pp.1053-1058, December 1972.

[10] Sha, L., Rajkumar, R., and Lehoczky, J.P.(1990). Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *In IEEE* Transactions on Computers, vol. 39, pp. 1175-1185.

[11] The AspectJ Primer, in WebPages at http://www.aspectj.org, The AspectJ Team.

[12] Wilner, D. (1997). " The Path Finder Invited Talk," *The 18th IEEE* Real-Time Systems Symposium, San Francisco, December, 1997.

PROCEEDINGS OF
THE 2006 INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING RESEARCH & PRACTICE
&
CONFERENCE ON PROGRAMMING LANGUAGES & COMPILERS

SERP'06

Volume II

Editors

Hamid R. Arabnia
Hassan Reza

Associate Editors

Nadim Asif, Lawrence Chung, Emanuel Grant
Ray Hashemi, Ashu M. G. Solo
Nary Subramanian

Las Vegas, Nevada, USA
June 26-29, 2006

This set of volumes contain papers presented at The 2006 International Conference on Software Engineering Research & Practice (SERP'06) and Conference on Programming Languages & Compilers (PLC'06). Their inclusion in this publication does not necessarily constitute endorsements by editors or by the publisher.

ISBN: 1-932415-90-4, 1-932415-91-2 (1-932415-92-0)
Printed in the United States of America

CSREA Press
U. S. A.

Foreword

It gives us great pleasure to introduce this collection of papers to be presented at the 2006 International Conference on Software Engineering Research & Practice (SERP'06) and Programming Languages & Compilers (PLC'06), June 26 through 29, 2006, at Monte Carlo Resort, Las Vegas, Nevada, USA.

The Academic Co-sponsors of this year's event include: Massachusetts Institute of Technology's (MIT) Media Lab (http://www.media.mit.edu/); Texas Advanced Computer Center (TACC) of University of Texas at Austin (http://www.tacc.utexas.edu/); Institute for Informatics Problems of Russian Academy of Sciences, Moscow, Russia (The IIP of Russian Academy of Sciences is co-sponsoring the ILINTEC Workshop and MLMTA Conference only - http://www.ipiran.ru/english/main.asp); and The Ohio Supercomputer Center (OSC is co-sponsoring the CIC Conference only - http://www.osc.edu).
Organizers and Co-sponsors at-large include: A number of university faculty members and their staff (names appear on the cover of proceedings); World Academy of Science (http://www.world-academy-of-science.org/worldcomp06/ws/index_html); and Computer Science Research, Education, & Applications Press.
Media Co-Sponsors include: HPC wire (http://www.hpcwire.com/); GRID today (http://www.gridtoday.com/); H2CM Hodges' Health (http://www.p-jones.demon.co.uk/); and Charter Cable Internet Access (http://www.ezisp.info/high-speed/cable-internet.html).
Other Co-Sponsors include: One Laptop Per Child Association (OLPC; http://www.laptop.org/); and International Technology Institute (http://www.itiworld.org/).
In addition to the above, several publishers of computer science and computer engineering books and journals, chapters and/or task forces of computer science associations/organizations from 12 countries, and developers of high-performance machines and systems have provided significant help in organizing the conference.

The program committee would like to thank all those who submitted papers for consideration. About 47% of the submissions were from outside the United States. Each submission was evaluated by two referees (except for papers that were submitted to chairs of sessions who were responsible for the evaluation of these papers.) The overall paper acceptance rate was about 34% (as of April 30, 2006).

We are very grateful to the many colleagues who helped in organizing the conference. In particular, we would like to thank the members of the SERP'06 and PLC'06 Program Committee who we hope will offer their help again in organizing the next year's conference (SERP'07). The SERP'06 and PLC'06 Program Committee members are:

Prof. Hamid R. Arabnia (Coordinator/Co-Chair), University of Georgia, USA (SERP/PLC)
Prof. Nadim Asif (Track Chair), University of Lahore, Lahore, Pakistan (SERP)
Rajesh K. Bhatia, Thapar Inst. of Engg. & Tech. (Deemed Univ.), Punjab, India (SERP)
Prof. William Cheng-Chung Chu, TungHai University, Taiwan (SERP)
Dr. Yoonsik Cheon, University of Texas at El Paso, El Paso, Texas, USA (SERP/PLC)
Prof. Lawrence Chung (Track Chair), Univ. of Texas at Dallas, Richardson, TX, USA (SERP)
Dr. Joseph Cote (Track Chair), CIO, Enterprise Architecture and Standards Division, Treasury Board of Canada, Ottawa, Canada (SERP)
Prof. Alfredo Cuzzocrea, University of Calabria, Cosenza, Italy (SERP)
Prof. Kevin Daimi, University of Detroit Mercy, Detroit, Michigan, USA (SERP)
Prof. Sergiu Dascalu, University of Nevada, Reno, Nevada, USA (SERP)

Prof. Virginia de Paula, Long Island University, Brooklyn Campus, New York, USA (SERP)
Dr. Jing Dong, University of Texas at Dallas, Richardson, Texas, USA (SERP)
Prof. Robert B. France, Colorado State University, Fort Collins, Colorado, USA (SERP)
Prof. Emanuel Grant (Track Chair), University of North Dakota, Grand Forks, ND, USA (SERP)
Dr. George A. Gravvanis, Democritus University of Thrace, Xanthi, Greece (SERP)
Dr. Volker Gruhn, University of Leipzig, Germany (SERP)
Dr. Zonghua Gu, Hong Kong University of Science & Technology, Kowloon, Hong Kong (SERP)
Dr. Jiang Guo, California State University Los Angeles, USA (SERP)
Dr. Timothy J. Harvey, Rice University, Houston, Texas, USA (PLC)
Prof. Ray Hashami, Armstrong Atlantic State University, Georgia, USA (SERP)
Dr. Hassan Hosseini, Cisco Systems, Ottawa, Ontario, Canada (SERP)
Drs Marc-Philippe Huget, University of Savoie, ESIA, Annecy, France (SERP)
Dr. Guohua Jin, Rice University, Houston, Texas, USA (PLC)
Dr. Carlos Juiz, Universitat de les Illes Balears, Despatx, Spain (SERP/PLC)
Dr. Osman Kandara, Southern University, Baton Rouge, Louisiana, USA (SERP)
Prof. Fereydoun Kazemian, Rochester Institute of Technology, New York, USA (SERP)
Dr. Anil Khatri, Bowie State University, Bowie, Maryland, USA (SERP)
Matthew Nicolas Kreeger, University of St. Andrews, UK and
nCipher Corporation Limited, Cambridge, England, UK (SERP)
Dr. Cyril S. Ku, William Paterson University, Wayne, New Jersey, USA (SERP)
Dr. Yan Luo, MEL, NIST, USA (SERP)
Sam Malek (PhD Student Member), University of Southern California, USA (SERP)
Dr. Johannes Mayer, University of Ulm, Ulm, Germany (SERP)
Subhas C. Misra (PhD Student Member), Carleton University, Canada (SERP/PLC)
Saeed Abbasi Moghaddam, Rayneel Corporation, Tehran, Iran (SERP)
Dr. Paniti Netinant, Bangkok University, Bangkok, Thailand (SERP)
Prof. Hassan Reza (Program Chair), Univ. of North Dakota, Grand Forks, USA (SERP/PLC)
Dr. Won W. Ro, California State University, Northridge, California, USA (PLC)
Ashu M. G. Solo, Maverick Technologies America Inc., Wilmington, Delaware, USA (SERP/PLC)
Dr. Ritu Soni, Guru Nanak Girls College and Kurukshertra University, India (SERP/PLC)
Dr. Omar Shatnawi, Al al-Bayt University, Mafraq, Jordan (SERP)
Dr. Nary Subramanian (Track Chair), University of Texas at Tyler, Tyler, Texas, USA (SERP)
Dr. Jiancun Wang, MonMouth University, USA (SERP)
Dr. Lingfeng Wang, Texas A&M University, College Station, Texas, USA (SERP)
Lizhe Wang, French National Research Inst. for CS & Control (INRIA), France (PLC)
Prof. Cong-Cong Xing (Track Chair), Nicholls State University, Louisiana, USA (PLC)
Dr. Sumanth Yenduri, University of Southern Mississippi, Gulfport, MS, USA (SERP/PLC)
Members of Task Force of FECS (Computer Science Curriculum Development)

We would also like to thank the followings: UCMSS (Universal Conference Management Systems & Support, San Diego, California, USA) for managing all aspects of the conference; Dr. Tim Field of APC for managing the printing of the proceedings; and the staff of Monte Carlo Resort in Las Vegas for the professional service they provided.
Last but not least, we would like to thank SERP'06 and PLC'06 Associate Editors, Drs. Nadim Asif, Lawrence Chung, Emanuel Grant, Ray Hashemi, Ashu M. G. Solo, and Nary Subramanian.

We present the proceedings of SERP/PLC'06.

Hamid R. Arabnia (General Chair) and Hassan Reza (Program Chair)

Contents

SESSION: SOFTWARE TESTING AND QUALITY ASSURANCE

A Framework for Automatic Testing of Industrial Controller Code 3
Dag Kristiansen, Karl–Petter Lindegaard

Agile Test–based Modeling 10
Bernhard Rumpe

Statistical Analysis and Enhancement of Random Testing Methods also under Constrained Resources 16
Johannes Mayer, Christoph Schneckenburger

DPTModel: The Defect Prevention and Traceability – Driven Model for Software Engineering 24
Jay Xiong, Jonathan Xiong

Selecting Effective Test Messages 31
Len Gebase, Roch Bertucat, Robert Snelick

Distributed Tool for Performance Testing 38
nenad stankovic

Test–bed for Verification and Validation Activities in Developing an Operations Support System 45
Dae–Woo Kim, Hyun–Min Lim, Sang–Kon Lee

Generation of Test Scenarios from Use Cases 52
Stephane Some

Restricted Adaptive Random Testing by Random Partitioning 59
Johannes Mayer

DPTMethodology: The Defect Prevention and Traceability – Driven Methodology for Software Engineering 66
Jay Xiong, Jonathan Xiong

The DPTSystem: The Defect Prevention and Traceability – Driven System for Software Engineering 73
Jay Xiong, Jonathan Xiong

Dynamically Generating Conformance Tests for Messaging Systems 80

Robert Snelick, Len Gebase, Sydney Henrard

Adapting Structural Testing to Functional Programming **86**
Manfred Widera

Critical Systems and Software Risk to Public Safety: Issues and Research Directions **93**
Shreedevi Inamdar, Hisham Haddad

Software Quality and Testing **100**
Hassan Pournaghshband, Asaleh Sharifi, Shahriar Movafaghi

A Method for Generating a Minimal Functional Set of Test–Cases for Software–Intensive Systems **106**
Joerg Gericke, Matthias Wiemann

Looking at Comparisons of Regression and Analogy–based Software Project Cost Prediction **113**
Carolyn Mair, Martin Shepperd

An Efficient Slicing Approach for Test Case Generation **119**
Durvasula V L N Somayajulu, Ajay Kumar Bothra, Prashant Kumar, Pratyush Pratyush

Impact of Using Test–Driven Development: A Case Study **126**
Sumanth Yenduri, Louise Perkins

Multi Dimension Quality Model of MAS **130**
Punam Bedi, Vibha Gaur

SESSION: SOFTWARE REUSE

Reusing Families Design **139**
Virginia de Paula

Reuse and Component Based Development (CBD) **146**
Rizwan Jameel

Retrieval of Most Relevant Reusable Component Using Genetic Algorithms **151**
Rajesh Bhatia, Mayank Dave, RC Joshi

A Reuse–Oriented Process Component Representation Framework **156**
Xiaohong Yang, Jing Lu, Ruzhi Xu, Guangfeng Pan, Jin Liu

Effective Reuse Procedure for Open Source Software **163**
Doo Yeon Kim, Jong Bae Kim, Sung Yul Rhew

Analysis of Object–Oriented Numerical Libraries 253
Kostas Zotos, George Stephanides

SESSION: SOFTWARE REQUIREMENT ANALYSIS

Requirements Engineering for E–Voting Systems 259
Kevin Daimi, Katherine Snyder, Robert James

Automatic Comprehension of Textual User Requirements and their Static and Dynamic Modeling 266
Olga Ormandjieva, Magda Ilieva

A Multi–Role Collaborative Method and Platform for Developing Software Requirements 274
Chin–Yi Tsai, Chua–Huang Huang

A Course Design and Implementation Experience on Agile Software Development Methodologies 281
Hongxing Lu, Xiaohong (Sophie) Wang

Software Development with Automatic Code Generation: Observations from Novice Developer Viewpoint 289
Farahzad Behi, Andrew Kornecki

The Factors of Software Systems that Contribute to Requirements Elicitation 296
Allison Scogin

SESSION: SOFTWARE ARCHITECTURE, DESIGN PATTERNS, AND FRAMEWORKS

EJB Performance Measurement Framework 303
Denis Gefter, Robert Chun

Analyzing Communication Patterns in Software Engineering Projects 310
H. Keith Edwards, Robert R. Puckett, Art Jolly

A SOA–Based IA Asset Management Architecture Using XML in E–Government 316
Namho Yoo, Hyeong–Ah Choi

OSGi Service Layer Enhancements 323
Nico Goeminne, Gregory De Jans, Jan Hollez, Bart Dhoedt, Filip De Turck, Frank Gielen

Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions 329

Andreas Schönberger, Guido Wirtz

Updating Software Architectures: A Style–Based Approach 336
Dalila Tamzalit, Mourad Oussalah, Olivier Le Goaer, Abdelhak–Djamel Seriai

Towards a Layered Architectural Design of a Persistence Framework 343
Sai Peck Lee, Tong Ming Lim, Ho–Jin Choi

Pattern–Oriented Design for Multi–Agent System: A Process Framework 350
Radziah Mohamad, Safaai Deris, Hany Ammar

The Role of Model–Oriented Software Architecture in Safety Engineering 357
Hassan Reza, Emanuel Grant

SESSION: DISTRIBUTED AND REAL TIME SYSTEMS

Application Platforms for Embedded Systems: Suitability of J2ME and .NET Compact Framework 367
Koen Victor, Yves Vandewoude, Yolande Berbers

Practical Technologies for Implementing Distributed Applications as Evolvable Software Systems (ESS) 375
Kendall Conrad, Vincent Schmidt

Comparison of Object Oriented Technology Automatic Codes Generating Tools for Safety Critical Real–time Software 382
Farahzad Behi, Daniel Penny III

Experiences in Distributed Software Development with Wiki 389
Khalid Al–asmari, Liguo Yu

Interlocutor System 394
Edson Barros, Roseli Lopes

Compositional Abstraction for Concurrent Programs 399
Junyan Qian, Baowen Xu

Transformation of the Ravenscar Profile Based Ada Real–time Application to the Verification–ready Statecharts : Reverse Engineering and Statemate approach 405
Chang Jin Kim, Jin–Young Choi

SESSION: SOFTWARE MAINTENANCE

An Effort Estimation by UML Points in Early Stage of Software Development 415

SangEun Kim, William Lively, Dick Simmons

Predicting Error Probability in the Eclipse Project 422
Raed Shatnawi, Wei Li, Huaming Zhang

Are the Changes Induced by the Defect Reports in the Open Source Software Maintenance? 429
Timo Koponen, Heli Lintula

A Model of Maintainability – Suggestion for Future Research 436
Mira Kajko–Mattsson, Gerardo Canfora, Dan Chiorean, Arie van Deursen, Tuomas Ihme, Meir M Lehman, Rupert Reiger, Torsten Engel, Josef Wernke

An Entropy–Based Approach to Assessing Object–Oriented Software Maintainability and Degradation –– A Method and Case Study 442
Hector Olague, Letha Etzkorn, Glenn Cox

A Software Traceability Validation For Change Impact Analysis of Object Oriented Software 453
Suhaimi Ibrahim, Norbik Idris, Malcolm Munro, Aziz Deraman

A Comparison of the Efficiencies of Code Inspections in Software Development and Maintenance 460
Liguo Yu, Robert Batzinger, Srini Ramaswamy

SESSION: SOFTWARE METRICS, CONFIGURATION AND PROJECT MANAGEMENT

Virus Removal Cost (VRC) Metric 469
Kuangnan Chang, Bobby Adkins

Towards an Extendable Software System for Information Integration 474
Paul Whitney, Christian Posse, Xingye Lei

A Workbench for Learning Enterprise Patterns 482
Paulo Sousa

Web Metrics: The way of improvement of quality of Non web–based systems 489
Shazia Arshad, Muhammad Shoaib, Abad Shah

Effect of Human Behavior in SDLC 495
Ashmeet Kaur, Ritu Soni

On the Role of Software Metrics in Applying Design Patterns 503

Niloofar Khedri, Masoud Rahgozar, Mahmoud Reza Hashemi

A Qualitative Study on PATT – A Project Assessment and Tracking Tool **510**
Fabio Marzullo, Geraldo Xexéo

Computations with Large Numbers **517**
Weihu Hong, Mingshen Wu

SESSION: UML, MDA, ...

On the Effectiveness of Source Code Transformations for Binary Obfuscation **527**
Matias Madou, Bertrand Anckaert, Bruno De Bus, Koen De Bosschere, Jan Cappaert, Bart Preneel

Model Driven Development with Interactive Use Cases and UML Models **534**
Paul Nguyen, Robert Chun

Medical Informatics and Medical Databases Approach in Modeling Healthcare Education System with Unified Modeling Language (UML) **541**
Anil Khatri, Azene Zenebe, David Anyiwo

Model Transformation Based on Meta Templates **547**
Hongming Liu, Lizhang Qin, Xiaoping Jia, Adam Steele

Using UML to Develop Verifiable Reactive Systems **554**
S. Fatemeh Alavizadeh, Marjan Sirjani

Developing Medical Information System with MDA and Web Services **562**
Simone A. B. Melo, Denivaldo Lopes, Zair Abdelouahab

UML Analysis Using State Diagrams **569**
Mohammad Alanazi, Jason Belt, David Gustafson

SESSION: COMPONENT ORIENTED SOFTWARE DEVELOPMENT

Plugin–Based Systems with Self–Organized Hierarchical Presentation **577**
Boto Bako, Andreas Borchert, Norbert Heidenbluth, Johannes Mayer

Algorithms for Optimally Tracing Time Critical Programs **585**
Sergej Alekseev

Assessment of Component–Based Systems with Distributed Object Technologies **592**
Jiang Guo, Yuehong Liao, Xichun Pei

A Java Instrumentation–based Analysis Approach for the Dynamic Behaviors of J2EE Applications 599
Yuehong Liao, Jiang Guo, Xichun Pei

SoCoEMo–COTS: A Software Economic Model for Commercial Off–the–shelf (COTS) Based Software Development 606
Sana Ben Abdallah Ben Lamine, Lamia Labed Jilani, Henda Hajjami Ben Ghezala

Conceptual Model for Integration of COTS Components 613
James Tollerson, Hisham Haddad

Process Component Plug–in Approach 620
Jin Myung Choi, Sung Yul Rhew

SESSION: FORMAL METHODS AND SPECIFICATION LANGUAGES, AND LANGUAGE DESIGN

Inspection of Concurrent Systems: Combining Tables, Theorem Proving and Model Checking 629
Vera Pantelic, Xiao–Hui Jin, Mark Lawford, David Parnas

On a GUI–based Editor for Z Specifications and its Applications 636
Hiroshi Ishikawa

A Formally Verified Geometric Modelling Core 643
Catherine Dubois, Jean–Marc Mota

Formal Verification of a Simple Automated Negotiation Protocol 650
George Dimitoglou, Okan Duzyol, Lawrence Owusu

Re–Engineering BLUE Financial System Using Round–Trip Engineering and Java Language Conversion Assistant 657
Salem Al–Agtash, Tamer Al–Dwairy, Adnan EL–Nasan, Bruce Mull, Mamdouh Barakat, Anas Shqair

A Base for Achieving Semantics for Prolog with Cut for Correct Observables 664
Lingzhong Zhao, Tianlong Gu, Junyan Qian, Guoyong Cai

Comparison of the Modeling Languages Alloy and UML 671
Yujing He

Supporting Separation of Concerns to Automation of Code Generation 678
Paniti Netinant

A Software Specification Language for RNA Pseudoknots 684
Keum-Young Sung

The Intelligent C Language Debugger 688
Ming Wang, Robert Chun

SESSION: CASE STUDY, USABILITY ENGINEERING, AND EDUCATION

Integrating User Centered Design in a Product Development Lifecycle Process: A Case Study 695
Karsten Nebe, Lennart Groetzbach, Ronald Hartwig

Learner-centered Technical Review in Programming Courses 702
Hongxing Lu, Xiaohong (Sophie) Wang

Development of an Ant Script Builder with Thought to Usability and Best Practices 710
Kalyana Gundamaraju, Michael Wainer

Service Learning, Software Engineering, and Hurricane Katrina – A Case Study 717
Donald Schwartz, Jonathan Spencer, Adam Huffman

Podcasts: Changing the Face of e-Learning 721
Saby Tavales, Sotirios Skevoulis

SESSION: SOFTWARE RELIABILITY MODELS AND RISK ANALYSIS

Supporting Software Fault Tree Analysis Using a Key Node Metric 727
Donald Needham, Sean Jones

Metrics in Risk Determination for Large-Scale Distributed Systems Maintenance 734
Maureen Raley, Letha Etzkorn

SESSION: 5TH INTERNATIONAL WORKSHOP ON SYSTEM/SOFTWARE ARCHITECTURES, IWSSA'06

Ontology-Driven Middleware for Next-Generation Train Backbones 743
Stijn Verstichel, Sofie Van Hoecke, Matthias Strobbe, Steven Van den Berghe, Filip De Turck, Frederik Vermeulen, Piet Demeester

System Modeling for Systematic Development of Groupware Applications 750
Manuel Noguera, Miguel González, José Luis Garrido, María Visitación Hurtado, María Luisa Rodríguez

Organization Modelling to Support Access Control for Collaborative Systems 757
Francisco Luis Gutierrez, Jose Luis Isla, Patricia Paderewski, Miguel Sanchez

An NFR–Based Framework for Aligning Software Architectures with System Architectures 764
Nary Subramanian, Lawrence Chung

Architecture–Centric Program Transformation for Distributed Systems 771
Chung–Horng Lung, Jianning Liu, Xiaoli Ling, Dan Jiang

Component–Aware System Architecting: A Software Interoperability 778
Weimin Ma, Kendra Cooper, Lawrence Chung

Position Paper: From Enterprise Architectures to Software Architectures using Requirements Engineering 785
Matthias Galster, Armin Eberlein, Mahmood Moussavi

Helping to Meet the Security Needs of Enterprises: Using FDAF to Build RBAC into Software Architectures 790
Lirong Dai, Kendra Cooper

Modeling of Evolution to Secure Application System: from Requirements Model to Software Architecture 797
Michael Shin

An Enterprise Architecture Process Model 804
François Coallier, Roger Champagne

A Model of Access Control for Data Materials Based on Ambient Calculus 811
Masaki Murakami

SESSION: PROCEEDINGS OF PLC'06 – DATA–FLOW ANALYSIS

A Fine–Grained Analysis of the Performance and Power Benefits of Compiler Optimizations for Embedded Devices 821
Jason W.A. Selby, Mark Giesbrecht

Complexity of Data Flow Analysis for Non–Separable Frameworks 828
Bageshri Sathe, Uday Khedker

SESSION: PROCEEDINGS OF PLC'06 – CODE OPTIMIZATION AND COMPILER GENERATION TECHNIQUES

Experience in Testing Compiler Optimizers Using Comparison Checking 837

Masataka Sassa, Daijiro Sudo

Deterministically Executing Concurrent Programs for Testing and Debugging 844
Steve MacDonald, Jun Chen, Diego Novillo

Compiler Generator for Creating MOF–compliant Source Code Models 851
Zoltán László, Tibor Sulyán

An Embedded Haskell Subset Implementation 858
Ian Lewis

User–Friendly Methodology for Automatic Exploration of Compiler Options: A Case Study on the Intel XScale Microarchitecture 866
Haiping Wu, Eunjung Park, Long Chen, Juan del Cuvillo, Guang R. Gao

A User–Friendly Methodology for Automatic Exploration of Compiler Options 873
Haiping Wu, Long Chen, Joseph Manzano, Guang R. Gao

SESSION: PROCEEDINGS OF PLC'06 – LOGIC, FUNCTIONAL, MODELING, NEW PROGRAMMING PARADIGMS

Implementation of Tag Representation in Prolog Virtual Machine 883
Guillaume Autran, Xining Li

XML Markup Languages Framework for Programming in 21st Century towards Managed Software Engineering 890
Khubaib Ahmed Qureshi, M Zeeshan Ali Ansari

Improved Graph–Based Lambda Lifting 896
Marco T. Morazan, Barbara Mucha

On Petri Nets and Predicate–Transition Nets 903
Andrea Röck, Ray Kresman

IncH: An Incremental Compiler for a Functional Language 910
James Gil de Lamadrid, Jill Zimmerman

Extensible and Adaptable System Software 916
Paniti Netinant

SESSION: PROCEEDINGS OF PLC'06 – REGISTER ALLOCATION, MEMORY MANAGEMENT, AND OO TECHNIQUES

Efficient and General On–Stack Replacement for Aggressive Program Specialization 925

Sunil Soman, Chandra Krintz

Java Virtual Machine: the key for accurated memory prefetching **933**
Yolanda Becerra, Jordi Garcia, Toni Cortes, Nacho Navarro

Evaluation Issues in Generic Programming with Inheritance and Templates in C++ **940**
Emil Vassev, Joey Paquet

String Concatenation Optimization on Java Bytecode **945**
Ye Henry Tian

Aspects of Memory Management in Java and C++ **952**
Emil Vassev, Joey Paquet

Supporting Separation of Concerns to Automation of Code Generation**

Paniti Netinant
Computer Science Department
Bangkok University
Bangkok, Thailand
paniti.n@bu.ac.th

Abstract

Aspect-oriented framework is a new paradigm that complements the aspect-oriented technology. The premise of aspect-oriented technology is the separation of concerns, where functional components are designed relatively in isolation of the non-functional components in order to avoid the code-tangling phenomena. In this paper we present a formal methodology that supports the aspectual behavioral modeling for concurrent software systems in order to aid the system designers in validating the design of a concurrent software system against its requirements and automating the implementation of these systems from their constructed models.

Keyword: Aspect Orientation, Code Automation, Framework, CASE

1. Introduction

Recent advances in the software technology have reaffirmed the need for building component-oriented software systems. Object-oriented technology has delivered sound results regard supporting code reuse, and design patterns have demonstrated that large-scale software systems can benefit the most from the best documented design structures.

Despite the tremendous success of object-oriented technology and design patterns to support code reusability and design reusability, little has been said regard the separation of concerns within the context of these technologies. Recent research [18, 19] in the separation of concerns has demonstrated the need for the separation of the functional requirements from the non-functional requirements in order to maximize code reuse and design reuse; non-functional requirements tend to cut-across the functional components. Till now aspect-oriented technology has offered very little to support the system engineer with a formal modeling technique in order to verify and validate the system design against its requirements. Modeling the behavior of concurrent software system based on aspect-orientation is an area of research that is in its infancy stages. The Unified Modeling Language, UML[1,2], has four major elements, use cases, class diagrams, sequence diagrams, and statecharts, that evolved over time in order to aid designers in the specification and modeling of object-oriented systems. Though use-cases are generally used as black-box behavioral specification, the other three elements are used as white-box behavioral specification. Uses cases are used mainly to describe the interaction between the systems and entities that may exist within its environment, i.e., Actors. Object-oriented technology generally uses class-diagram to show the relationship between classes and the interaction that occur between classes, collaboration diagrams show the sequence in which the interactions between classes occur, and statechart diagram to model the class's behavior as a state machine; sequence diagrams describe the inter-object interactions and statecharts describes the intra-object interactions.

Our research is bout bringing a formal methodology to automate the implementation and validation of concurrent software systems based on aspect-orientation. An important element of a sound behavioral modeling approach is a rigorous methodology that ensures the semantics of the constructed model based on requirements. These models can be used to generate code and discover conflicting requirements. Concurrent software systems are composed of functional requirements and concurrency requirements, like synchronization constraints, and scheduling policies. Aspect-oriented technology is an ideal solution for modeling such systems, since the distinguishing between aspects and functional requirements is at the heart of such technology. Since system requirements may describe the interactions between classes or the sequence of interactions between classes, a formal notation is required to model such

* This research has been supported by Thailand Research Funding (TRF) organization and Bangkok University. The contract is MRG4780168.

requirements and UML's class diagram and sequence diagrams is an example of such notation. UML uses the use cases in order to cross-reference requirements and the sequence diagrams that trigger the use cases. What motivated our work is the need for a complete end-to-end requirement modeling technique that will guide the designer in the design of concurrent software systems and automate their implementations. Behavioral modeling based on requirements is the first step toward formal design methodology based on aspect-orientation.

2. System Software and Separation of Concerns

Modeling concurrent software systems with sequence diagrams and class diagrams will not be enough to achieve a full implementation of the system behavior [7]. Sequence diagrams will describe the inter object behavior, it will help in describing scenarios for the system, how objects will interact with each other, but will not allow zooming into the object itself to describe its behavior, for that we need Statecharts. It has been argued in [7] that MSC and UML's sequence diagrams can be used to specify requirements and the desired behavior for a system under development, but they can't be used to implement the desired behavior for a system under development. Statecharts, on the other hand provide the behavior of a particular object including various states that an object can enter into over its life cycle. As stated in [Harel] Statecharts are mainly used to describe intra-object behavior versus inter object behavior which can be specified using MSCs or Sequence Diagrams. Avery distinguishable characteristic of Statecharts from MSCs or Sequence Diagrams is the fact that Statecharts are part of the system model, which means they can be used to generate code.

State charts [8, 9, 10, and 11] show the states of an object within a given context, the events that causes an object to go from one state to another, and the actions that can occur as a result of a state transition. Events could be guarded by conditions, which must evaluate to true before a transition takes place. During the object's life cycle an object move from one state to another, and produce some actions as a result of this transition, this is how the behavior of an object should be described and Statecharts does that. Modeling concurrent systems based on aspect-orientation is a new approach that has many benefits: reusability, verification and validation, and automate implementation. UML/Statecharts are used mainly to model the internal behavior of concurrent objects, though this formalism of modeling doesn't support aspect orientation directly. To support aspect-oriented modeling within the scope of Statecharts, we extend the Statecharts in order to allow the explicit representation of aspect within the system models. Within concurrent objects, state transitions may occur if a method is invoked or a timer expired. Associating aspects that are evaluated when an object method is invoked is a troublesome for system modeler and developer, since the specification and constrains of these aspects are state dependents. For example, in the case where we have a bounded buffer with certain capacity and two methods put and get, we shall allow only put method to proceed when the buffer is an empty state and allow either put or get when we are in a partial state. Synchronization constraints and Scheduling specifications are the main aspects that influence the execution of the invoked methods based on object states. Being able to express the concurrent object behavior as StateChart and augment that by notation that allows the modeler to express aspects explicitly within the models is a step forward to automate the implementation and ease the ease the verification of such objects.

3. Aspect Orientation Approach

Figure 1 shows the model that captures the dynamic behavior for a bounded buffer, where there are two methods: put and get and three states where the concurrent object could be in. As we stated earlier current specification of stats charts doesn't support aspect-oriented modeling. To support aspect-orientation within the context of state charts, we need to provide a mechanism by which the modeler can express these aspects. In state charts, state transition may occur as a result of events, method invocation, or timer expiration. Therefore support to aspect modeling shall take into consideration that aspects are associated with methods, transitions, but not with states. Though as we will see shortly, automating the process of mapping the model into workable implementation will be state driven.

Figure 2 shows the aspects that are associated with the method invocations in the different states. Although aspects will have the same names that are associated with their perspective methods in the different states, their implementation may differ and the distinction between the different aspect implementation will be identified based on object states.

4. Aspect-Oriented Framework

The framework consists of four components comprising the architecture of the framework.

- Each functional object (component) provides its services (methods) stripped of any aspectual properties (for example, no synchronization is included in Buffer objects).
- A proxy object intercepts called methods and transfers the calls to the AspectModerator.

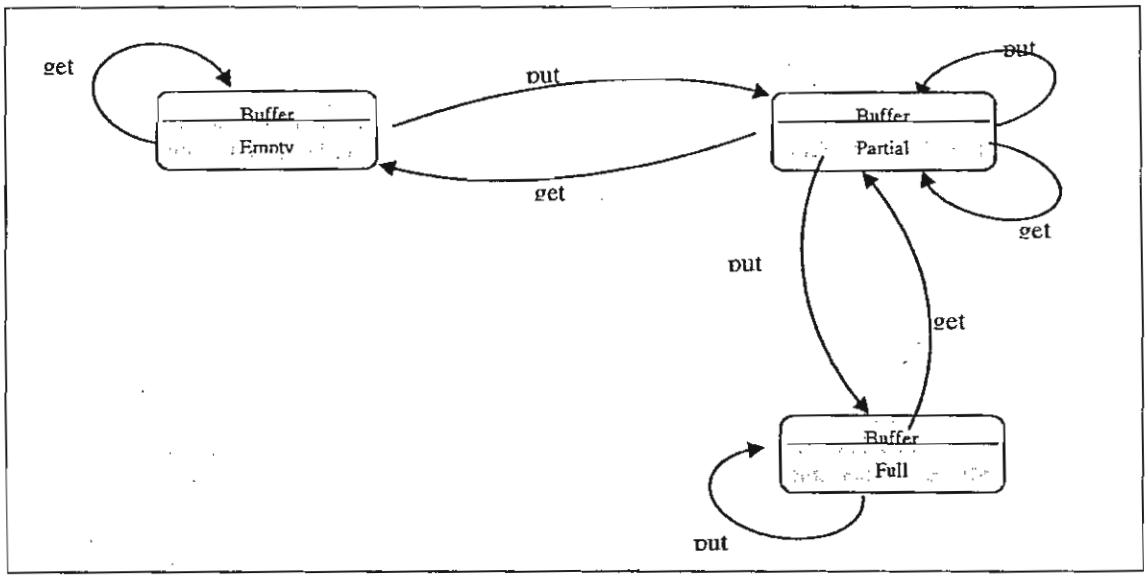


Figure 1: StateChart representation for the Bounded Buffer Concurrent Object Model

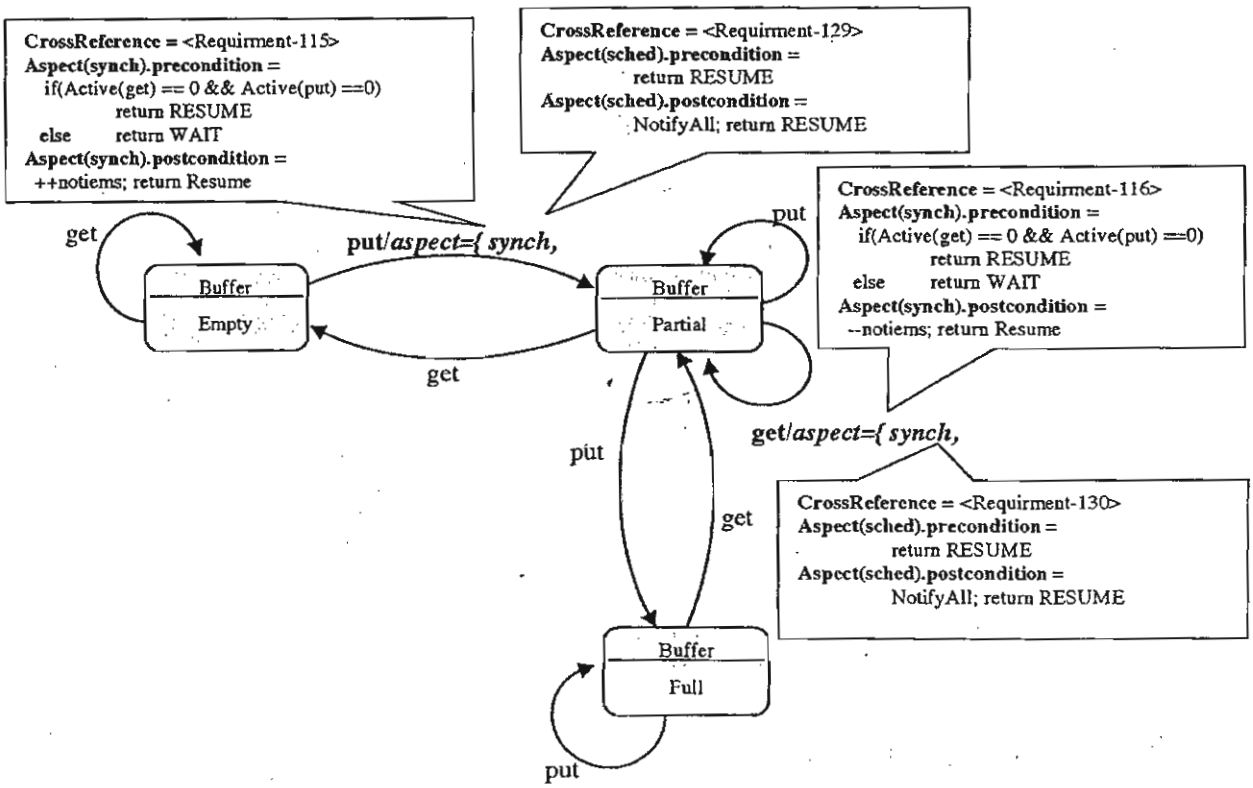


Figure 2: Bounded Buffer Behavioral Model

- An AspectModerator object consists of the rules and strategies needed to bind aspects at runtime. Aspects are selected from the AspectBank. The AspectModerator orders the execution of aspects. The order of execution can be static or dynamic. Then, each precondition will be checked whether it is satisfied or not.
- An AspectBank object consists of aspect objects that implement different policies of a variety of aspects.

This section presents the design and development of aspect-oriented framework. The model is presented to demonstrate horizontal composition of the framework. The system service must be implemented as a Component class. The system aspectual property (SystemAspect class) must be derived from the SystemAbstractAspect interface to implement the required behavior of a system aspectual property. A SystemAspectFactory consists of many system aspectual properties such as synchronization, tracing, logging, and reliability. The SystemAspectFactory, derived from the SystemAbstractAspectFactory interface, is known as an aspect bank.

During runtime, each SystemAspectFactory will be associated with one SystemAspect. The AspectModerator class must be derived from the AspectModerator interface to implement the required behavior. The following points are important about the aspect-oriented framework:

- A base layer framework is an implementation of an underlying system.
- An application layer framework is an implementation of application software over the system software represented by a base layer framework.
- A client object requests a service through a ProxyObject object of a framework.
- A functional component is implemented as a Component class without any aspectual property.
- A SystemAspectFactory object consists of various SystemAspect objects. A SystemAspect object is controlled by a SystemAspectFactory object.
- Each system aspectual property must be implemented as a SystemAspect object.
- Each crosscutting between Component object and a SystemAspect object must be defined in AspectModerator object as joinpoints in a Pointcut method.
- A client requests a service by sending a message to a ProxyObject object. The ProxyObject object changes the request to a specific pointcut method, and forwards it to the AspectModerator object.

The Proxy class is responsible for intercepting and forwarding the message sent from Client object to request a service. The Proxy class must implement the behavior of intercepting a service request. A client object of an aspect-oriented framework must request a service by calling the call() method. A call() method consists of at least two parameters: object name provided a service and a service requested to serve. The first parameter is of type string, and the second is type of string as well. The ProxyObject class will forward a request to the AspectModerator object by calling a PointCut() method. A PointCut() method must have the same number parameters and the same parameter type as the call() method.

The SystemAspectFactor class must be derived from the SystemAspectFactoryAbstract interface to implement the required behavior. The AspectModerator class is responsible for composing the functional components and the system aspectual property into a service request. The AspectModerator class acts like a coordinator between functional components and system aspectual properties, when and where system aspectual properties will be composed into a functional component. The composition of system aspectual properties and functional components must be guided and defined as PointCut() method. Each PointCut() method must have at least two parameters: component name and service name (methods of the component) that will be composed. The first parameter is of type string, and the second is type of string as well.

The AspectModerator class will create the SystemAspectFactory object. The SystemAspectFactory object can support either static or dynamic aspects at runtime. The attachImple() method is used to associate a system aspectual property of a SystemAspectFactory object. The AspectModerator class will be associated with functional components and system aspectual properties that will be composed. The PointCut() method will define join points between functional components and system aspectual property. Currently, the PointCut() method uses if...then...else... statements to define joinpoints. The synchronization aspect property crosscuts both read and write services. It crosscuts the before and after execution of read and write services. A tracing property crosscuts both read and write services. It only crosscuts the after execution of read and write services.

The SystemAspectFactor class must be derived from the SystemAspectFactoryAbstract interface to implement the required behavior. The SystemAspectFactory class provides a dynamic binding of variety system aspectual properties. It focuses on the interface of the system aspectual property. Each system aspectual property must be derived from the SystemAspectAbstract interface to implement the required behavior. Implementation of a system aspectual property is implemented in the SystemAspect class. Each system aspectual property can define before(), after(), and precondition() methods depending on its needs.

The AspectModerator class operates composition between system aspectual properties and functional components using a composition rule defined by join points of a pointcut. The AspectModerator class performs composi-

tion rules by sending AspectFactory messages. Messages sending causes polymorphism. The implementation of AspectFactory uses bridge patterns. A message finds the correct member object of the AspectFactory, and invokes that object. With polymorphism calls, AspectModerator requires less information about each SystemAspect, so the AspectModerator only needs to have the right SystemAspect interface.

The abstract aspectual class defines a SystemAbstractAspect interface that controls the implementation of an aspectual property class. This class is implemented using the concrete classes of aspectual properties, which implement the virtual functions before() and after(). The AspectModerator creates instances of an aspectual property, which requires composing a requested service. If an aspectual property crosscuts more than one method in the same component, it must have a parameter ServiceName identifying what it should be done for each method. If an aspectual property crosscuts more than one component, it must have two parameters: ServiceName and ComponentName identify what it should be done for each method of each component.

5. Automating Code Generation from Models

For each aspect associated with operation, the tool will generate a class for that aspect. Figure 3 shows the code that will be generated to represent the synchronization aspect class for put method. It is important to notice that aspects are can be listed next to each method invocation in each of the specified object states. The aspects are linked to template that the modeler will fill out whenever the aspect is created, though in figure 3 the aspect ***callouts*** are shown for illustration purposes. An important feature in our approach is the ability to trace requirements into implementation. In our approach requirements can be cross-referenced in the model itself, where requirement numbers can be associated with transitions. The order of listing the aspects with each method invocation is relevant, since that the order that these aspects will be evaluated upon method invocations. So in our example in figure 3, the synch aspect is evaluated before the sched aspect. Unlike other approaches [Rahpsody] where the aspectual code is where aspectual code is inter-mixed with the functional code, our approach relies on aspect orientation to separate concerns and by the same token generate clean classes that purely represent the aspectual code.

```
Private class putSynchAspect implements Aspect {
....
precondition() {
if(state == Empty)
        if(Active(get) == 0 && Active(put) ==0)
                return RESUME
        else    return WAIT

if(state == Partial)
        if(Active(get) == 0 && Active(put) ==0)
                return RESUME
        else    return WAIT

if(state == Full)
        return WAIT

}

postcondition() {
if(state = empty) state = Partial;
if(state = empty)
        if(noitems < bSize –2) state = Partial;
        else    state = FULL

 ++notiems;
return Resume
}

}


Private class putSchedAspect implements Aspect {
....
precondition() {
if(state == Empty)
        if(Active(get) == 0 && Active(put) ==0)
                return RESUME
        else    return WAIT
}

postcondition() {
state = Partial;
 --notiems;
```

Figure 3: The Generated Aspect Classes

6. Related Work

Behavioral modeling is getting more attention from researchers as well as practitioners since system engineers and designers describe the system requirements based on the expected system behavior. Needless to say that most testing is based black-box testing.

Modeling concurrent system behavior based on StateCharts [7] is the first to address the issue of verification and validation of concurrent object-oriented systems. Though the approach suffers from code-tangling phenomenon where concurrency code is intermixed with the functionality code. UMLAUT is a recent approach based on aspect-oriented modeling that addresses the separation of concerns early in the design phase, though this approach didn't address the intra-object interactions. Our approach addresses aspect-oriented modeling for concurrent systems based on extension to statecharts in order to automate the implementation for such systems.

7. Conclusion

Aspect-oriented programming is the next wave of development for software systems. This paradigm stresses the separation of the ascpetual code from the functionality code in order to maximize code reusability and minimize changes that are due to code-tangling. Behavioral modeling based on statecharts are getting more popular since it addresses modeling the system behavior early in the design stage and automate the implementation of such systems. Our approach is an aspect-oriented modeling technique that extends statecharts in order to allow the explicit representation of aspects in the behavioral models, and automate the implementation of these systems from their perspective models. Inheritance of aspects with the system models is an area that we believe require further research.

8. References

[1] UML Summery V1.1, OMG, ad/97-08-03, www.rational.com/UML.

[2] UML Syntax and Semantics Guide V1.1, OMG, ad/97-08-03, www.rational.com/UML.

[3] Junichi Suzuki, Yoshikazu Yamamotto, "Extending UML with Aspects:Aspect support in the design phase". The 3rd AOP Workshop at ECOOP 1999.

[4] Siobhan Clarket, William Harison, Ossher, Tarr, "Separation Concerns throughout the Development lifecycle". The 3rd AOP Workshop at ECOOP 1999

[5] Siobhan Clarket "Extending UML Metamodel for Design Composition". The 3rd AOP Workshop at ECOOP 1999

[6] Juan, Papathomas, Murillo, Sanchez, "Coordinating Concurrent Objects: How to deal with the coordination aspect?". The 1st AOP Workshop at ECOOP 1997.

[7] David Harel "From Play-In Scenarios To Code: An Achievable Dream". IEEE Computer, to appear. Preliminary version in Proc. Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science, Vol. 1783, Springer-Verlag, March 2000, pp. 22-34.

[8] W. Ho, F. Pennaneac'h , J. Jezequel, and N. Plouzeau, "Aspect-Oriented Design with the UML*".

[9] D Harel and M. Politi, "Modeling Reactive systems with StateCharts". McGraw-Hill, New York, 1994.

[10] Bruce Powel Dpouglass, "UML Statecharts". ESP Jan-1999. I-Logix

[11] Harel, Daveid "Statecharts: a visual formalism for complex systems". Science of Computer Programming. Vol. 8 (1987),P. 231

[12] Lodewijk Bergmans and Mehmet Aksit. "Composing Software from Multiple Concerns: A Model and Composition Anomalies". ICSE 2000 2nd Workshop on Multidimensional Separation of Concerns.

[13] A.Bader, C. A. Constantinides, T. Elrad, T. Fuller, P. Netinant. "Building Reusable Concurrent Software Systems". International Conference on Parallel and Distributed Techniques and Applications (PDPTA'2000) special session on Distributed Objects in Computational Science. June 26 - 29, 2000. Las Vegas, Nevada, USA.

[14] Constantinos A. Constantinides and Tzilla Elrad. "On the Requirements for Concurrent Software Architectures to Support Advanced Separation of Concerns". Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2000) Workshop on Advanced Separation of Concerns in Object-Oriented Systems. Minneapolis, Minnesota, USA. October 16, 2000.

[15] Constantinos A. Constantinides, Atef Bader and Tzilla Elrad. "Separation of Concerns in the Design of Concurrent Software Systems". The 14th European Conference on Object-Oriented Programming (ECOOP 2000) Workshop on Aspects and Dimensions of Concerns. Sophia Antipolis and Cannes, France, June 11-12, 2000.

[16] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, Reading, MA,1995.

[17] Ralph E. Johnson. "Frameworks = (Components + Patterns)". In Communications of the ACM. Vol. 40. No. 10. October 1997, pp. 39-42.

[18] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. "Aspect-Oriented Programming". Proceedings of ECOOP '97. LNCS 1241. Springer-Verlag, pp. 220-242.

[19] Kim Mens, Cristina Lopes, Badir Tekinerdogan, and Gregor Kiczales. "Aspect-Oriented Programming". Report of the ECOOP '97 Workshop for Aspect-Oriented Programming.

[20] Sinan Si Alhir "Extending UML"

[21] Sinan Si Alhir "What is UML"

[22] Sinan Si Alhir "UML Extension Mechanisms"

[23] Rhapsody from I-Logix www.ilogix.com, Rhapsody Guid.

EXTENSIBLE AND ADAPTABLE SYSTEM SOFTWARE*

Paniti Netinant
Computer Science Department
Bangkok University
Bangkok, Thailand
paniti.n@bu.ac.th

ABSTRACT

Concurrent real-time software systems are vulnerable to performance saturation and reliability concerns due to environmental influences. Building intelligent concurrent systems that are able to adapt to environmental changes and reconfigure themselves is the key to avoiding performance degradation of concurrent real-time software systems and ensuring the liveness property of such systems. In this paper we present a machine learning-based approach that addresses the design of agent-based intelligent concurrent software systems in order to ensure the reliability and performance properties for such systems. Although reliability and performance are conflicting requirements in most cases, we will show how to use an aspect-oriented technology by which these requirements can be designed, implemented, reused, and replaced in isolation from each other. The performance and reliability of the software system can be reasoned about by intelligent agents who can direct the system to reconfigure itself in order to adapt to the environment changes. The agents rely on the data-mining techniques to discover patterns of performance degradation or imminent signals of reliability violation and to predict policies that cope best with the environmental fluctuations.

KEYWORDS: Aspect-Oriented Framework, Agents, Reusability, Extensibility, and System Software

1. INTRODUCTION

Concurrent real-time systems are designed mainly in order to ensure performance and the stringent reliability requirements, sometimes referred to as quality properties. Until recently, the design and development of these systems have inter-mixed the quality code with the functionality code for such systems; the quality properties cut across the functional components. This cut-across phenomenon [7, 12, 13, and 14] generally breaks the component model, and makes it hard to design, and reuse, especially when the functional and non-functional requirements change. Generally requirement changes force reengineering for these systems. Generally, the functional components for these systems are stable. On the other hand, the quality requirements are volatile and reactive to the environment changes. Another issue that necessitates engineering reconfigurability when designing these systems is the make It possible to build these systems so that and real-time properties can be replaced while the systems are running, in order to adapt to environmental changes.

Building intelligent software systems that have open architectures, which support reconfigurability, is essential for concurrent real-time applications by which their well-being and performance are heavily dependent on their capability to cope with the environment fluctuations. The Mars pathfinder problem [19, 20] is a classical example of such systems where conflict of interest between performance and liveness properties has forced system reset that was due to a priority inversion problem. The Mars Pathfinder spacecraft's engineers were aware of the priority inheritance solution for such problems but they preferred not to use it [27], since it may cause performance degradation for the spacecraft. The lack of software adaptability hooks is the main reason that NASA system engineers chose not to deploy priority inheritance mechanism in order to avoid the system resetting.

Aspect-Oriented Programming (AOP) [1, 2, 7, 8, and 9] is a new programming paradigm that attempts to separate the functional components from the interaction components (aspects). Aspects are defined as properties that cut across groups of functional components. While these aspects can be thought about and analyzed relatively separately from the basic functionality, at the implementation level they must be combined together. Programming concerns manually into the system's functionality using current component-oriented languages results in aspects being tangle throughout the code. This code tangling makes the source code difficult to develop, understand and evolve because it destroys modularity and reduces

This research has been supported by Thailand Research Funding (TRF) organization and Bangkok University. The contract MRG4780168.

software quality [8]. In this paper we show how to deploy aspect-oriented technology, which provides an architectural support for the design and development of intelligent concurrent systems. We show how the aspect code can be isolated from the functional components that otherwise would be intermingled with the code of the functional components. Isolating the functional components from the non-functional components, the aspect code, has many attractive benefits: first and foremost it promotes reusability for the functional classes and the aspect classes. It also simplifies the design of complex systems, since the interaction code is separated from the functional code. Our approach is a step toward building reconfigurable intelligent systems and improves the software quality as it complements the object-oriented and component-oriented technologies with a set of design principles in order to engineer adaptability into software systems.

In our approach, assembling the intelligent concurrent systems from both functional and non-functional components through the use of aspect-oriented technology supports both static and dynamic adaptability when building the intelligent concurrent systems.

Recent advances in information technology have demonstrated that there are numerous issues that may affect the quality of service for software systems and that the only way to improve the quality of these services is by using software agents that monitor, advise, and react based on the environment changes. A software agent is an autonomous software component that can react to and interact with its environment. An agent is autonomous, since it runs in its own thread of control, and reactive, because of its capability to respond to incoming messages.

The agent-based approach is still in its infancy, although it is getting more popular as more IT applications are using this approach. Enterprise applications, B2B applications, Personal Agents, Information filtering, Information monitoring, and Interface agents / personal assistants just to name a few. For example, in information monitoring, activities are dependent on the timely notification of changes in the environment or in the data sources. Agents are very useful for monitoring different data sources for specific data. Agents can be dispatched to remote locations to monitor data sources. An interface agent is a program that is able to operate within a user interface and actively assist the user in operating the interface and manipulating the underlying system. MS-Office assistants are an example of this category.

Our experience shows that agent-based approaches are a key component in building an intelligent concurrent system. They complement the aspect-oriented and object-oriented technologies, while an aspect-oriented solution complements object-oriented technology to solve the code-tangling phenomenon and improve code reusability. The agent-based approach complements these technologies in order to support dynamic adaptability and ensure quality of services.

Frameworks capture design decisions that are common to applications in certain domains. Generally, frameworks emphasize design reuse over code reuse, although a framework may have concrete subclasses that can be used immediately. In this paper we present a framework that can be used to build intelligent concurrent systems. The key contribution of this work is to show how to deploy aspect-orientation in the design of these systems so that system requirements that may have an impact on performance, reliability, and security are isolated from the functional components, and intelligent agents are deployed to watch each one of these aspects. As we will be described in the subsequent sections, aspect-orientation within our framework helps to engineer reconfigurability into the intelligent systems such that policies can be altered, reused, or replaced without halting the running system. We will also show how to design the intelligent agents based on data mining techniques in order to audit and guide the real-time system from its own training data and to update its knowledge base in real-time.

2. AGENTS FOR ASPECT-ORIENTED CONCURRENT SYSTEMS

Concurrent object-oriented open software systems are composed of functional requirements and concurrency requirements. Mixing the functional code and concurrency may impede code reuse; this has been documented in the literature as the inheritance anomaly problem. Solutions for the inheritance anomaly problem vary from domain specific languages (ABCL) to framework based solutions [4]. Framework solutions are preferred over domain specific languages, since these frameworks are based on common object-oriented languages and require less time to learn and deploy. Despite the success of the framework based approaches, support of static and dynamic adaptability for concurrent software systems has not been addressed in a formal way.

Recent aspect-oriented approaches [4] have provided an elegant solution to support the static adaptability aspect for concurrent software systems, though the dynamic adaptability aspect has been left un-addressed. Aspect-oriented technology complements the object-oriented technology in order to avoid so-called code-tangling phenomena and support static adaptability for concurrent open software systems; Adaptability [5] is an important factor that enables software systems to evolve in order to meet future requirements. Reflective approaches [8, 9] offer solutions to support the dynamic adaptability aspect for open software systems, though reflection-based

solutions have hard-coded decision processes that react and adapt to environment changes in an intelligent way.

Agent-oriented technology is getting more popular as more industrial applications start to deploy this approach. Agents are needed mainly to deal with uncertainty and react to environment changes and they are very useful to monitor systems resources and notify the interested parties to change their behavior to cope with the environment changes. Methods to engineer intelligence and machine learning within agents vary from data mining [17] techniques to q-learning [10]. In [17], the authors presented an approach based on the data-mining technique to discover patterns of behavior and network intrusion detection. And in [10, 11] the authors have demonstrated how q-routing algorithms can be used to discover best routes in a highly congested network.

Our approach integrates agent-oriented technology and aspect-oriented technology to build intelligent concurrent software systems [3,6]; systems that run within an uncertain environment and require the capability of altering their components and policies during run time. Figure 1 shows our integrated view of these technologies. Our approach delivers a framework solution for building these systems. The agents are needed to assist in the decision making process for reconfiguring the software system during run time. Aspect-orientation techniques are used since they isolate the functional components from the aspectual components like performance and reliability.

Figure 1. A concurrent object as a cluster of components and aspects within the aspect moderator framework.

The aspect-oriented approach [4] has demonstrated its effectiveness in building concurrent object-oriented systems, where the concurrency aspects, like synchronization constraints and scheduling policies, are isolated from the functional components. This approach helps in building a stable software system that can easily adapt to meet future requirements and react to environment changes.

The approach stated in [4] did resolve the static adaptability problem, but did not address the dynamic adaptability aspect for building intelligent concurrent software systems. Our research has revealed the need to have intelligent components, agents that can aid the concurrent software system in the decision-making process to reconfigure itself in order to adapt to environmental changes. The agents in our approach deploy the data mining technique and the Bayesian algorithm to monitor system resources and predicate patterns of performance degradation or imminent signals of reliability violation, and offer advice to react to these changes. Dealing with environmental uncertainty is the key challenge for concurrent real-time systems. Environmental uncertainty may have an impact on performance, reliability, throughput, or quality of service guarantees. One way to monitor system resources and environment concept drifts is by gathering data about usage of systems resources. In our framework we apply Bayesian algorithms and online learning techniques to predict environment changes and adapt to these changes. For example, when a buffer is more than half-full, and the ratio of the number of waiting puts to the number of waiting gets is 1 to 10, we may still prefer put over get, if though the immediate preference shall be given to get, the historical data demonstrates that whenever the buffer is more than half full, the number of waiting gets was substantially greater than waiting puts. This distinction between immediate preferences and desires and long-term well-being objectives has been fully discussed and advocated in the artificial intelligence discipline. Our research has revealed that such distinction is extremely important in building open software systems that operate in volatile environments, where system resources can go through over utilized and underutilized cycles.

Through the support of agents, our framework can be used to build concurrent open software systems that can dynamically adapt to environmental changes and deal with uncertainty. Agents are used to monitor certain aspects of the software systems; scheduling is an example of such aspects. Agents have a knowledge base that they consult to predict the system behavior, and update their knowledge base to keep pace with current environment settings

3. ASPECT-ORIENTED FRAMEWORK

A sequential object is comprised of functionality control and shared data. Access to this shared data is controlled by synchronization and scheduling abstractions. Synchronization controls enable or

disable method invocations for selection. The synchronization abstraction is composed of guards and post-actions. During the Precondition phase, guards will validate the synchronization conditions. In the Notification phase, post-actions will update the synchronization variables. The scheduling abstraction allows the specification of scheduling restrictions and terminate-actions. At the Precondition phase, scheduling restrictions use scheduling counters to form the scheduling condition for each method. At the Notification phase, terminate actions update the scheduling counters. During the Precondition phase, the synchronization constraints of the invoked method are evaluated. If the current synchronization condition evaluates to RESUME the scheduling constraints are then evaluated. After executing the Precondition phase, the moderator will activate the method in the sequential object. During Notification, synchronization variables and scheduling counters are updated upon method completion. The aspect moderator object coordinates functional and aspectual behavior, by handling their interdependencies. We stress the fact that the activation order of the aspects is the most important part in order to verify the semantics of the system. Synchronization has to be verified before scheduling. A possible reverse in the order of activation may violate the semantics. There are other issues that might also be involved. If authentication is introduced to a shared object for example, it must be handling before synchronization.

A major component of quality in software is reliability: a system's ability to perform its job according to the specification (correctness) and to handle abnormal situations (robustness). In [24] introduces the concept of "design by contract" in the context of the Eiffel programming language [25]. Under this theory, a software system is viewed as a set of communicating components whose interaction is based on precisely defined specifications of the mutual obligations known as contracts. These contracts govern the interaction of the element with the rest of the world. The importance of assertions is also stressed in [14] where it is described how the absence of specifications caused the disaster associated with the European Ariane 5 launcher. The aspect moderator framework adopts this approach in a different context: defining assertions (preconditions and postconditions) as a set of design principles. Meyer argues that assertion monitoring yields to a productive approach to debugging, testing and quality assurance, in which the search for errors is not blind but based on consistency conditions provided by the developers themselves. As a result, reliability should be a built-in component in software development, not an afterthought. None of Java, Ada [DoD80] or CORBA [OMG98] has any built-in support for design by contract.

In [14] the authors argue that without specification it is probably safer to redo rather than to reuse. Another important issue is the one of the verification of components and aspects in isolation from each other. One must be able to test the functionality of a component as well as being able to test that an aspect will align nicely with the functional component. Otherwise, there can be no guarantee that components and aspects will co-operate. In other words, one must test and verify the collaboration of components and aspects. This would constitute an important phase in the design process.

3.1 ADAPTABILITY

There is a general feeling that OOP promotes reuse and expandability by its very nature. We argue that this is a misconception as none of these issues is enforced. Rather, a software system must be specifically designed for reuse and expandability. Adaptability is an important quality factor in software systems and the issue of it being explicitly engineered into a system is stressed in [12]. Incremental adaptability means coping with changing requirements without modifying previously defined software components. The conventional object-oriented model supports adaptability through composition, encapsulation, message passing and inheritance mechanisms. In general, lack of support of dynamic adaptability might lead to re-engineering the whole software system. In [30] it is argued that concurrent OO languages do not provide enough support for the development of true adaptable software either because aspects are mixed in the functional components, or because once components are woven the resulting piece of software is too rigid to be adapted or reconfigured at run-time.

The general architecture of the framework allows reusability and ensures adaptability of components and aspects as both are designed relatively separately from each other. The aspect moderator is a design pattern that hooks components and aspects together, defining their semantic interaction. The use of design patterns in order to provide axes of adaptability is suggested in [12]. One of the advantages of the aspect moderator framework is that if a new aspect of concern would have to be added to the system, we do not need to modify the moderator class. We can simply create a new class to inherit and re-define it, and reuse it for a new behavior. The inherited class can handle all previous aspects, together with the newly added aspect. Adaptability is also applied to components. The aspect-moderator framework does not require some new syntactic structure for the representation of new aspects, but simply a new class for the new aspect. This technique makes it easy for an existing aspect to be removed from the overall system. In this framework, the moderator object has the capability to activate or drop aspects on the fly. Further, the semantic interaction between components and aspects in the

framework is defined by a set of principles. Part of this semantic interaction is the order of activation of the aspects thus providing a criterion for aspect ordering. The order of execution can also be altered on the fly. This concept is not feasible with automatic weaver technologies. In this framework, components and aspects are designed relatively separately and they remain separate entities that may access each other freely without code transformation. In fact, functional components do not need to know about the aspect components in advance (before run-time) but only after an aspect has been created and registered by the moderator class. As a result, components and aspects discover each other at run-time if necessary. The interaction of newly added aspects with the rest of the system is handling in a similar manner as the implementer must specify the contract that binds a new aspect to the rest of the system rather than having to re-engineer the whole system. On the other hand, automatic weavers must rely on language constructs that are hard coded into aspect code to provide the contact (join) points.

In [22] the authors stressed the importance of aspect manifestation in every stage of development. The issue that in some cases aspects should remain run-time entities was also discussed in [5, 15] also stressed this issue by arguing that much like conditional compilation, aspects must be woven to the program on-demand. In technologies that rely on automatic weaving, aspects manifest in the model and in the program code, but neither in object code (byte code in the case of Java) nor in executable (binary) code. [7] argues that with static weaving it might be impossible to adapting or replacing aspects dynamically. The framework manages to achieve the manifestation of aspects at run-time. We argue that is important that in order to achieve maximum flexibility a framework must provide for dynamic aspect evolution and ideally support both static and dynamic behavior. As an example, an aspect such as synchronization can be statically dealt with.

3.2 COMPOSITION OF ASPECTS

In ESP [8] and the Adaptive Arena [1] the functional part of a system is separated from the synchronization code, but it still remains in the same class. The separation of functional and aspectual code in the aspect moderator framework results in program code that is more modular. Furthermore, the framework follows a general-purpose approach in order to achieve composition of concerns. This way, it is not confined to certain aspects but can address a number of aspects. It is also language neutral. With the exception of AspectJ, current technologies are confined in domain specific languages. We introduce the concept of an aspect bank, where the moderator of a cluster initially needs to collect and register all the required aspects from. The aspect bank provides a hierarchical two-dimensional composition of the system in terms of aspects and components.

3.3 RELATED WORK

There are quite a few approaches that attempt to build intelligent systems. But what is of interest for us are the approaches that allow us to build an intelligent concurrent real-time system in such a way that reconfigurability and reusability of the software system are supported. JAM [16] is an agent-based approach for detecting network intrusion. JAM is implemented by the JAVA language and it deploys the data mining techniques in the rule classifications. JAM has been designed mainly to address the security aspects of software systems; other aspects like performance and reliability are hard to reason about. Q-routing [17, 18] is another recent example that addresses routing in a highly congested network. This approach is faster in general than approaches that rely on the data mining techniques, but it suffers from the fact that history data is not fully used on the decision making process. The work presented in [15] is an approach for load balancing a network of computers, this approach is mainly based on classification rules, but it does not consider reconfigurability and reusability in the design and development of concurrent real-time systems.

4. CONCLUSION

Agent-based software systems are the next wave in the software engineering discipline. Recent advances in software technology have stressed the need to build intelligent open software systems in order to build dynamically reconfigurable software systems that can deal with environment uncertainty and predict usage patterns for system resources. A key factor for building highly reconfigurable software systems is to identify and isolate the aspectual code from the functional code in order to maximize reusability and facilitate reconfigurability. Aspect-oriented programming is an emerging programming technique that makes it possible to engineer the reconfigurability of software systems. We use software agents in our framework in order to support the decision-making processes for such systems. Our framework based approach integrates aspect-oriented technology and agent-orientation in order to support the dynamic adaptability aspect for intelligent concurrent software systems, where agents can be asked to monitor system properties and react to undesirable events by reconfiguring the software without halting the running system. The Mars Pathfinder resetting problem was due to hard-coded decisions. The Pathfinder software system was not able to reason about the environment

and itself. We can build intelligent software systems based on agents and aspect-orientation, such that the decision making process that copes with the environment fluctuations is isolated from the core functional components. Agents can be used to alter the bias of the system, whenever conflicts arise; between performance and reliability.

5. REFERENCES

[1] Bardou, D. (1998). Roles, Subjects, and Aspects. How Do They Relate? Position paper, *ECOOP '98* Workshop on Aspect-Oriented Programming , pp.55-59.

[2] Berger L., Dery M. and Fornarino M. (1998). Interactions Between Objects: An Aspect of Object-Oriented Languages. Position paper, *ECOOP '98* Workshop on Aspect-Oriented Programming.

[3] Callsen, C. J., and Agha, G. A. (1994). Open Heterogeneous Computing in Actor Space. *Journal of Parallel and Distributed Computing*, 1994, pp. 289--300.

[4] Constantinides C., Bader A., Elrad T. (1999)., A Framework to Address a Two-dimensional Composition of Concerns Position paper, *Object-Oriented Programming: Systems Languages and Applications (OOPSLA '99)* First Workshop on Multidimensional Separation of Concerns in Object-Oriented Systems. Denver, Colorado (USA). November 1, 1999.

[5] Fayad, M., and Cline, M. (1996). Aspects of Software Adaptability, *Communications of the ACM*, 39(10), 1996, 58-59.

[6] Kiczales, G., Lamping J., Mendhekar, A., Maeda,C, Lopes C., Loingtier,J-M, and Irwin J. (1997). Aspect-Oriented Programming. *In Proceedings of ECOOP '97.* LNCS 1241. Springer-Verlag, pp. 220-242.

[7] Lopes, C. and Kiczales, G. (1998). Recent Developments in AspectJ. *Position paper ECOOP '98* Workshop on Aspect-Oriented Programming.

[8] Pryor, J.and Bastán, N. (1999). A Reflective Architecture for the Support of Aspect-Oriented Programming in Smalltalk. *Position paper, ECOOP'99* Workshop on Aspect-Oriented Programming.

[9] Bershad, B. Savage, S. Pardyak, P. Sirer, G. Fiuczynski, M. Becker, D. Eggers, S. and Chamber, C.1999: Extensibility, Safety, and performance in the SPIN Operating System *ECOOP'98.*

[10] Boyan J. and Littman M.(1995). A Distributed Reinforcement Learning Scheme for Network Routing. *Technical Report CMU=CS-93-165.*

[11] Kiczales, G. Lamping, J., Mendhekar, A. Maeda, C. Lopes, C. Loingtier, J-M., and Irwin J. (1997). "Aspect-Oriented Programming". *PARC Technical Report,* SPL97-008P9710042 pp 69-71.

[12] Kiczales, G., Lamping, J., Mendhekar, A. Maeda, C., Lopes, C., Loingtier,J-M., and Irwin, J.(1998). " Aspect-Oriented Programming". In M.Askit and S. Matsuoka, editors, *Proceedings of the 12th European Conference on Object-Oriented Programming,*. ECCOP'98, Springer Verlag, Berlin pp 88-95.

[13] Kiczales, G. (1996), *Aspect-Oriented Programming: A Position Paper from the Xerox PARC Aspect-Oriented Programming Project,* Xerox PARC, Palo, Alto, CA.

[14] Kubat, M. (1998), "*A Machine Learning-Based Approach to Load Balancing in Computer Networks,*" Cybernetics and Systems Journal, 23, 1992, pp. 389-400.

[15] Stolfo, S., Fan, W., Lee, W. Prodromidis, A., and Chan, P. Proc. DARPA Information Survivability Conference and Exposition, IEEE Computer Press, p. II 130-144, 2000.

[16] Boyan, J. A., and Littman, M. L. (1994). *Packet routing in dynamically changing networks: A reinforcement learning approach.* In Cowan, J. D.; Tesauro, G.; and Alspector, J., eds., Advances In Neural Information Processing Systems 6. Morgan Kaufmann Publishers.

[17] Boyan, J. A., and Littman, M. L. (1994). *Packet Routing in Dinamically Changing Networks: A reinforcement learning approach,* In Advances in Neural Information Processing Systems 6 (NIPS6), 1994, 671—678.

[18] Diekmann, R., Frommer, A., and Monien, B. (1999). Efficient Schemes for Nearest Neighbor Load Balancing. *Technical report, Dept. Maths. Comp. Sci., Univ.* Paderborn, Furstenallee 11, D-33102 Paderborn, Germany.

[19] Sha, L., Rajkumar, R., and Lehoczky, J.P.(1990). Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *In IEEE* Transactions on Computers, vol. 39, pp. 1175-1185.

[20] Wilner, D. (1997). " The Path Finder Invited Talk," *The 18th IEEE* Real-Time Systems Symposium, San Francisco, December, 1997.

ECTI-CON 2006

Proceedings of the 2006 Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI) International Conference

Wednesday May 10 – Saturday May 13, 2006

Ubonburi Hotel, Ubon Ratchathani, THAILAND

Organized by

Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI) Association

ECTI-CON 2006

Proceedings of the 2006 Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI) International Conference

Wednesday May 10 – Saturday May 13, 2006

Ubonburi Hotel, Ubon Ratchathani, THAILAND

Organized by

Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI) Association

---

Steering Committee

Pansak Siriruchatapong (NECTEC), Chair
Wanlop Surakampontorn (KMITL)
Sawasd Tantaratana (SIIT)
Sawat Tantiphanwadi (NSTDA)
Akachai Sang-in (CMU)
Sinchai Kamolphiwong (PSU)
Weerapant Musigasarn (PSU)
Chidchanok Lursinsap (CU)
Kobchai Dejhan (KMITL)
Somchai Chatratana (KMITNB)
Somsak Choomchuay (KMITL), Secretary

International Advisory Committee

Akinori Nishihara, (Tokyo Tech., Japan)
Hara Shinji (U. of Tokyo, Japan)
Luigi Benedicenti, (U. of Regina, Canada)
Rolf H. Jansen (Aachen U., Germany)
Yong-Hwan Lee (SNU, Korea)
Narong Yoothanom (SPU)

Organizing Committee

General Chair
Wanlop Surakampontorn (KMITL)

Vice chairs
Monai Krairiksh (KMITL)

International coordination chair
Saykhong Saynasine (NUOL,Laos)
Hang Chan (RUPP,Cambodia)
Nicholas Shuley (UQ,Australia)
Pung Keng (NUS,Singapore)

Technical Program Co-Chairs
Athikom Roeksabutr (MUT)
Chaiwut Chat-uthai (KMITL)
Bundit Thipakorn (KMUTT)
Jun-ichi Takada (TIT,Japan)
Prabhas Chongsatitwattana (CU)
Vutipong Areekul (KU)
Waree Kongprawechnon (SIIT)
Jitkasem Ngamnil (MUT)

Special Session Chairs
Kazushi Nakano (UEC,Japan)
Prayoot Akkaraekthalin (KMITNB)
Chuwong Phongcharoenpanich (KMITL)

Local Arrangement Chairs
Werachet Khan-ngenn (KMITL)
Mongkol Pasuyatanont (UBU)
Rungrangsee Vibulchai (UNC)
Surajate On-rit (UBRU)
Apirat Siritaratiwai (KKU)
Phaophak Sirisuk (MUT)
Yingrak Auttawaitkul (RTU)
Prasit Surasil (UVC)

Publication Chairs
Apirat Siritaratiwai (KKU)
Raungrong Suleesathira (KMUTT)
Danai Torrungrueng (AUST)

Publicity Chairs
Pinit Kumhom (KMUTT)
Anantawat Kunakorn (KMITL)

Exhibition Chair
Keattisak Sripimanwat (NECTEC)
Sdhabhon Bhokha (UBU)
Petmanee Viriyasudphong (UNC)
Supachate Innet (UTCC)
Denchai Worasawate (KU)
Supaporn Buphaprohm (UVC)

Finance Chairs
Banlue Srisuchinwong (SIIT)
Vutipong Areekul (KU)

General Secretary
Chalie Charoenlarpnopparut (SIIT)
Apinunt Thanachayanont (KMITL)

Co-Sponsored by
National Electronics and Computer Technology Center (NECTEC)
National Science and Technology Development Agency (NSTDA)

Technical collaboration with
IEEE Communications Society, Thailand Chapter
IEEE Circuits and Systems Society, Thailand Chapter
IEEE MTT/AP/ED, Thailand Chapter
IEEE Laser and Electro-Optics Society, Thailand Chapter

Table of Contents

- Welcome Message from Steering Committee Chair i
- Welcome Message from General Chair ii
- Welcome Message from Technical Program Chair iii
- List of Reviewers iv
- Symposium Schedule at a Glance v
- ECTI-CON 2006 Session Schedule vi
- Keynote Speech: Thursday, May 11, 2006 viii
- Technical Program Contents xxix
- Technical Papers: Thursday, May 11, 2006 1
- Author Index A-1

Message

from

Chairman of Steering Committee

The third ECTI-CON is organized as international conference at Ubonburi Hotel in Ubon Ratchathani province Thailand on May 10-13, 2006. The steering committee of ECTI-CON has determined Ubon Ratchathani province as the venue for the third ECTI-CON in the beginning of 2005, since we have a policy to expand opportunity for researchers not only in nationwide of Thailand, but also neighboring countries in Indo-China region such as Cambodia, Laos, and Vietnam.

Ubon Ratchathani is an ancient town with 4,000 year-old culture located on the Maekhong river bank in the center of triangle among Laos, Cambodia, and Thailand. It has the biggest population and area in the south of north eastern part of Thailand, and can be counted as center of this region. This is the main reason that the committee decided to move venue to this province in 2006. Hopefully, the third ECTI-CON 2006 works as research gateway to this region, and many researchers in this region become more active, especially in the ECTI-CON in the future.

In the next ECTI-CON in 2007, the steering committee has a concept to move the venue back to the center of Thailand nearby Bangkok, and go to remote region again in 2008.

On behalf of the steering committee of ECTI-CON, I would like to show my appreciations against the excellent organizing works performed by all of ECTI-CON 2006 organizing committee members and staff, and also excellent research results presented by researchers around the world. Hopefully, the participants of ECTI-CON 2006 will enjoy presentation, discussion, banquet, local food, local culture, and tour. Finally, I look forward to meeting you again at the ECTI-CON 2007 next year.

Pansak siriruchatapong
Chairman of ECTI-CON Steering Committee

Message

from

General Committee Chair

It is indeed my great honor to cordially welcome all the participants to the 2006 Electrical/Electronics, Computer, Telecommunications, and Information Technology Conference (ECTI 2006), held on May 10-13 2006, in Ubonburi Hotel and Resort, Ubon Ratchathani province, Thailand. This is the third annual conference in the series, where the first ECTI Conf kicks off in the year 2004 as one of the major activities of the ECTI Association. It should be noticed that this is the first time that we extend the conference to the place that is not in the popular places like Pataya, Chaingmai and Phuket. However, we still got a very good response.

The objective of the conference is to annually bring together researchers from Thailand as well as other parts of the world to discuss and exchange experiences with the aim to stimulate and enhance the research and development in the areas that are related to Electrical/Electronics, Communications and Information Technologies. It is also to provide a forum for the discussion of original works, new ideas and new recent advances in the areas.

In the capacity of the Organizing Committee Chair, I would like to express sincere appreciation to the significant contributions and efforts by organizing committee members, especially the General Secretary: Assoc. Prof. Dr. Kosin Chamnongthai, the Technical Program Chair: Assoc Prof. Dr. Athikom Roeksabutr and the Local Arrangement Chair, Assoc. Prof. Dr. Werachet Khan-ngenn, which make this conference a great success. In addition, the valuable contributions from the authors that submitted technical papers for review, attendees, technical program committee members, speakers, and session chairs are gratefully acknowledged. We do hope that, you participate in the conference with pleasure and find a good opportunity to meet, to exchange ideas and to make research contacts and collaboration.

Wanlop Surakampontorn
General Committee Chair ECTI-CON 2006

Message

from

Technical Program Chair

Welcome to ECTI-CON 2006 - the third annual international conference organized by Electrical Engineering/Electronics, Computer, Telecommunications and Information (ECTI) Technology association of Thailand. This happens to be the first international conference in Ubon Ratchathani Province, an ancient town with 4000 year old culture on the Maekhong river bank.

This year conference offers an outstanding program in 47 sessions for about 192 contributed papers, being accepted from 230 submitted papers from more 8 countries. The reviewers, who are the experts in the particular fields, were working very hard in voluntary to select those quality contributed papers. In addition, 28 invited papers are also included in special sessions, whose area is currently in the hot issue.

"**Technology for Life**" is the theme for ECTI-CON 2006. I believe that you will find it true after attending technical sessions through the conference.

All keynote speakers have been honorably invited to give speech on the topics that should encourage research community in Thailand as well as illustrate the research scenery of Thai neighbors.

I would like to thank keynote speakers, and all the technical program committee members and chairs who voluntarily invest their own time inviting speakers, selecting papers, and arranging such the impressive conference. Of course, sincere thanks must finally go to all authors, without whom the conference would not occur, who make contribution of their papers to the conference.

Thank for your participation in ECTI-CON 2006. I strongly believe this is a good opportunity to share knowledge and experiences among participants. Please have a great time and use this opportunity to meet more people and make yourself known by exchanging experiences and both technical and non-technical information.

Athikom Roeksabutr
Technical Program Chair of ECTI-CON 2006

List of Reviewers

We would like to express special thanks to the following individual and anonymous reviewers for their effort in the review process of ECTI-CON 2006:

Akinori Nishihara
Amorntep Jirattitichareon
Anantawat Kunakorn
Andrew Davison
Anuwat Jangwanitlert
Apichai Bhatranand
Apichan Kanjanavapastit
Apinunt Thanachayanont
Apirat Siritaratiwat
Apisak Worapishet
Athikom Roeksabutr
Atsushi Takahashi
Bongkarn Homnan
Boonserm Kijsirikul
Bundhit Eua-arporn
Bundit Thipakorn
Chaiwut Chat-Uthai
Chalie Charoenlarpnopparut
Chanchai Laohapengsang
Chanin Bunlaksananusorn
Chanjira Sinthanayothin
Chaodit Aswakul
Chiranut Sa-ngiamsak
Chokchai Sangdao
Chutham Sawigun
Chuwong Phongcharoenpanich
Danai Torrungrueng
Daranee Hormdee
David Banjerdpongchai
Denchai Worasawate
Ekachai Leelarasmee
Hiroshi Tamura
Issarachai Ngamroo
Itsda Boonyaroonate
Jatuporn Chinrungrueng
Jitkasame Ngarmnil
Jun-ichi Takada
Kasin Vichienchom
Kazushi Nakano
Kitti Attakitmongcol
Kittipong Tonmitr
Kohji Higuchi
Komsak Meksamoot
Kosin Chamnongthai
Kriangkrai Sooksood
Lunchakorn Wuttisittikulkij
Mitchai Chongcheawchamnan
Monai Krairiksh
Namkhun Srisanit
Narumol Kiatwarin
Nimit Chomnawang
Nipapon Siripon
Nipon Theera-Umpon
Nongluk Covavisaruch
Nontawat Chuladaycha
Pakorn Kaewtrakulpong
Panumas Khumsat
Panuthat Boonpramuk
Parnjit Damrongkulkamjorn
Pathomthat Chiradeja
Peerapol Yuvapoositanon
Peng Hin Lee
Phakphoom Boonyanant
Phaophak Sirisuk
Pichai Aree
Pinit Thepsatorn
Piyasawat Navaratana Na Ayudhy
Prabhas Chongsatitwattana
Prajuab Pawarangkoon
Prawit Chumchu
Prayoot Akkaraekthalin
Saliltip Sinthusonthishat
Sanpachai Huvanandana
Sansanee Auephanwiriyakul
Santi Asawasripongtorn
Sanya Mitaim
Sawasd Tantaratana
Sawat Tantiphanwadi
Siripun Thongchai
Siriroj Sirisukprasert
Somboon Sangwongwanich
Somchat Jiriwibhakorn
Sompob Polmai
Somporn Sirisumrannukul
Somying Thainimit
Songsak Chusanapipat
Stanislav Makhanov
Surapan Airphaiboon
Tanee Demeechai
Tawan Phurat
Taweedej Sirithanapipat
Taworn B
Techaumnat Boonchai
Teerasit Kasetkasem
Thanatchai Kulworawanichpong
Thawatchai Meeteevarunyoo
Thumrongrat Amornraksa
Tiparatana Wongcharoen
Toshiaki Kondo
Toshihisa Tanaka
Varakorn Kasemsuwan
Vutipong Areekul
Wanlop Surakampontorn
Warakorn Charoensuk
Waree Kongprawechnon
Wichian Chutimaskul
Worapong Tangsrirat
Yongyuth Permpoontanalarp
Yoshikazu Miyanaga

Symposium Schedule at a Glance

Wednesday, May 10, 2006

16:00-18:30	Registration
18:00-21:00	Welcome Party

Thursday, May 11, 2006

08:00-08:45	Registration
08:45-09:00	Opening ceremony
09:00-09:40	Keynote speech: "Trend of HDD Research in Thailand" by Mr. Brent L. Bargmann, Executive Vice President, Seagate Technology (Thailand)
09:40-10:20	Keynote speech: "A Review of Uncooperative Target Identification Using UWB Resonance Based Radar Techniques" by Prof. Dr. Nicholas Shuley, University of Queensland, Australia
10:20-10:30	Coffee Break
10:30-11:00	Keynote speech: "Status and Trend of ICT in Lao PDR" by Dr. Saykhong Saynasine, National University of Laos, Laos
11:00-11:30	Keynote speech: "ICT in Cambodia" by Mr. Hang Chan Thon, Royal University of Phnom Penh, Russian Federation Blvd, Phnom Penh, Cambodia
11:30-12:00	Keynote speech: "The Power Development Plan of Thailand" by Dr. Suthep ChimKlay (EGAT) IEEE Thailand
12:10-13:00	Lunch Break
13:00-14:40	Technical Sessions
14:40-15:00	Coffee Break
15:00-17:00	Technical Sessions
18:00-21:00	Conference Banquet

Friday, May 12, 2006

08:30-10:10	Technical Sessions
10:10-10:30	Coffee Break
10:30-12:10	Technical Sessions
12:10-13:00	Lunch Break
13:00-14:40	Technical Sessions
14:40-15:00	Coffee Break
15:00-16:40	Technical Sessions
17:00-18:00	Closing Ceremony

ECTI-CON 2006 Session Schedule

Time	Room	
		Wednesday, May 10, 2006
16:00-18:30		Registration
18:00-21:00		Welcome Party
		Thursday, May 11th, 2006
08:00-08:45		Registration
08:45-09:00		Opening ceremony
09:00-09:40		Keynote speech: "Trend of HDD Research in Thailand" by Mr. Brent L. Bargmann, Executive Vice President, Seagate Technology (Thailand)
09:40-10:20		Keynote speech: "A Review of Uncooperative Target Identification Using UWB Resonance Based Radar Techniques" by Prof.Dr. Nicholas Shuley, University of Queensland, Australia
10:20-10:30		Break
10:30-11:00		Keynote speech: "Status and Trend of ICT in Lao PDR" by Dr. Saykhong Saynasine, National University of Laos, Laos
11:00-11:30		Keynote speech: "ICT in Cambodia" by Mr. Hang Chan Thon, Royal University of Phnom Penh, Russian Federation Blvd, Phnom Penh, Cambodia
11:30-12:00		Keynote speech: "The Power Development Plan of Thailand" by Dr. Suthep ChimKlay (EGAT) IEEE Thailand
12:10-13:00		Lunch Break
	Session	TPM1-1 / TPM1-2 / TPM1-3 / TPM1-4 / TPM1-5 / TPM1-6 / TPM1-7 / TPM1-8
		VLSI Design / VLSI Design / Computer System and Applications / Signal Processing I / Electric machinery / Special session on Microwave Technology / Control Theory and Application I / Optical Technology and Communications
	Chairperson	Dr. Sufian M. Mitani (Multimedia University, Malaysia) / Assoc. Prof. Dr. Ekachai Leelarasamee (CU) / Dr. Panita Pongpaibool (NECTEC) / Dr. Chalie Charoenlarpnopparut (SIIT) / Asst. Prof. Dr. Supat Kittiratsatcha (KMITL) / Asst. Prof. Sithiporn Kerdsuwang (KMITNB) / Prof. Kou Yamada (Gunma University, Japan) / Assoc. Prof. Dr. Prayoot Akkaraekthalin (KMITNB)
13:00-13:20		TPM1-1-1 / TPM1-2-1 / TPM1-3-1 / TPM1-4-1 / TPM1-5-1 / TPM1-6-1 / TPM1-7-1 / TPM1-8-1
13:20-13:40		TPM1-1-2 / TPM1-2-2 / TPM1-3-2 / TPM1-4-2 / TPM1-5-2 / TPM1-6-2 / TPM1-7-2 / TPM1-8-2
13:40-14:00		TPM1-1-3 / TPM1-2-3 / TPM1-3-3 / TPM1-4-3 / TPM1-5-3 / TPM1-6-3 / TPM1-7-3 / TPM1-8-3
14:00-14:20		TPM1-1-4 / TPM1-2-4 / TPM1-3-4 / TPM1-4-4 / TPM1-5-4 / TPM1-6-4 / TPM1-7-4 / TPM1-8-4
14:20-14:40		TPM1-1-5 / TPM1-2-5 / TPM1-3-5 / TPM1-4-5 / TPM1-5-5 / TPM1-6-5 / TPM1-7-5 / TPM1-8-5
14:40-15:00		Break
	Session	TPM2-1 / TPM2-2 / TPM2-3 / TPM2-4 / TPM2-5 / TPM2-6 / TPM2-7 / TPM2-8
		CMOS Technology / Special session on Cryptography, Smart Card and RFID / Modelling and Simulation / Signal Processing II / Energy and power systems I / Special session on 3G Technology / Control Device and Intrument / Microwave and Antenna
	Chairperson	Dr. Chiranuch Sa-Ngiumsak (KKU) / Dr.Chaichana Mitrpant (NECTEC) / Asst. Prof. Dr. Paniti Netinant (BU) / Dr. Chai Wuttiwatchai (NECTEC) / Assoc. Prof. Dr. Anantawat Kunakorn (KMITL) / Asst. Prof. Dr. Chuwong Phongchareonpanich (KMITL) / Asst. Prof. Pakorn Kaewtrakulpong (KMUTT) / Asst. Prof. Dr. Danai Torrungrueng (AU)
15:00-15:20		TPM2-1-1 / TPM2-2-1 / TPM2-3-1 / TPM2-4-1 / TPM2-5-1 / TPM2-6-1 / TPM2-7-1 / TPM2-8-1
15:20-15:40		TPM2-1-2 / TPM2-2-2 / TPM2-3-2 / TPM2-4-2 / TPM2-5-2 / TPM2-6-2 / TPM2-7-2 / TPM2-8-2
15:40-16:00		TPM2-1-3 / TPM2-2-3 / TPM2-3-3 / TPM2-4-3 / TPM2-5-3 / TPM2-6-3 / TPM2-7-3 / TPM2-8-3
16:00-16:20		TPM2-1-4 / TPM2-2-4 / TPM2-3-4 / TPM2-4-4 / TPM2-5-4 / TPM2-6-4 / TPM2-7-4 / TPM2-8-4
16:20-16:40		TPM2-1-5 / TPM2-2-5 / TPM2-3-5 / TPM2-4-5 / TPM2-5-5 / TPM2-6-5 / TPM2-7-5 / TPM2-8-5
16:40-17:00		TPM2-1-6 / TPM2-3-6 / TPM2-5-6 / TPM2-6-6 / TPM2-8-6
18:00-21:00		Conference Banquet

ECTI-CON 2006 Session Schedule

(Continued)

Friday, May 12th, 2006

	Session	FAM1-1	FAM1-2	FAM1-3	FAM1-4	FAM1-5	FAM1-6	FAM1-7	FAM1-8
		Devices and Modelling	MEMS and nanoelectronic devices	Network Protocol and Technique	High voltage and insulation	Energy and power systems II	Special session on Wireless Networks	Mobile Communications	Antenna and Propagation
	Chairperson	Asst. Prof. Dr. Pirat Kumhom (KMUTT)	Dr. Songphol Kanjanachchai (CU)	Prof. Tomoaki Sato (Hirosaki University, Japan)	Assoc. Prof. Dr. Chaiwut Chat-Uthai (KMITL)	Assoc. Prof. Dr. Werachet Khan-Ngern (KMITL)	Dr. Chavalit Srisathapornphat (KU)	Sathaporn Promwong (KMITL)	Dr.Denchai Worasawate (KU)
08:30-08:50		-	-	FAM1-3-1	FAM1-4-1	FAM1-5-1	FAM1-6-1	FAM1-7-1	FAM1-8-1
08:50-09:10		FAM1-1-1	FAM1-2-1	FAM1-3-2	FAM1-4-2	FAM1-5-2	FAM1-6-2	FAM1-7-2	FAM1-8-2
09:10-09:30		FAM1-1-2	FAM1-2-2	FAM1-3-3	FAM1-4-3	FAM1-5-3	FAM1-6-3	FAM1-7-3	FAM1-8-3
09:30-09:50		FAM1-1-3	FAM1-2-3	FAM1-3-4	FAM1-4-4	FAM1-5-4	FAM1-6-4	FAM1-7-4	FAM1-8-4
09:50-10:10		FAM1-1-4	FAM1-2-4	FAM1-3-5	FAM1-4-5	FAM1-5-5	FAM1-6-5	FAM1-7-5	FAM1-8-5
10:10-10:30		Break							
	Session	FAM2-1	FAM2-2	FAM2-3	FAM2-4	FAM2-5	FAM2-6	FAM2-7	FAM2-8
		Analog Circuits and Filters I	Genetic Algorithms	Multimedia Technology	Power System Protection and Transmission	Power Engineering Applications	Special session on Control in Powertronics and Mechatronics	Wireless Communications	Ultra-Wideband Communications
	Chairperson	Assoc. Prof. Dr. Worapong Tangsrirat (KMITL)	Dr. Low Sew Ming, (Moansh University Malaysia)	Dr. Yongyut Permpoontanalarp (KMUTT)	Dr. Teratam Bunyagul (KMITNB)	Asst. Prof. Dr. Itsda Boonyaroonate (KMUTT)	Prof. Kazushi Nakano (UEC, Japan)	Asst. Prof. Dr. Phichet Moungnoul (KMITL)	Prof. Jun-ichi Takada (TIT, Japan)
10:30-10:50		FAM2-1-1	FAM2-2-1	FAM2-3-1	FAM2-4-1	FAM2-5-1	FAM2-6-1	FAM2-7-1	FAM2-8-1
10:50-11:10		FAM2-1-2	FAM2-2-2	FAM2-3-2	FAM2-4-2	FAM2-5-2	FAM2-6-2	FAM2-7-2	FAM2-8-2
11:10-11:30		FAM2-1-3	FAM2-2-3	FAM2-3-3	FAM2-4-3	FAM2-5-3	FAM2-6-3	FAM2-7-3	FAM2-8-3
11:30-11:50		FAM2-1-4	FAM2-2-4	FAM2-3-4	FAM2-4-4	FAM2-5-4	FAM2-6-4	FAM2-7-4	FAM2-8-4
11:50-12:10		FAM2-1-5	FAM2-2-5	FAM2-3-5	FAM2-4-5	FAM2-5-5	FAM2-6-5	FAM2-7-5	FAM2-8-5
12:10-13:00		Lunch Break							
	Session	FPM1-1	FPM1-2	FPM1-3	FPM1-4	FPM1-5	FPM1-6	FPM1-7	FPM1-8
		Analog Circuits and Filters II	Enginering Educations, Measurement, and Management	Artificial Intelligence	Image Processing I	Power Control System	Special session on Antenna Technology	Fuzzy and Neural Network	Communication Theory and Applications
	Chairperson	Dr. Sanya Khunkhao (SU)	Dr. Kittipong Meesawat (KKU)	Dr. Prapas Chongsatitwattana (CU)	Dr. Matthew Dailey (AIT)	Asst. Prof. Phaophak Sirisuk (MUT)	Prof. Monai Krairiksh (KMITL)	Dr. Wudhichai Assawinchaichote (KMUTT)	Dr. Tuptim Angkew (CU)
13:00-13:20		FPM1-1-1	FPM1-2-1	FPM1-3-1	FPM1-4-1	FPM1-5-1	FPM1-6-1	FPM1-7-1	FPM1-8-1
13:20-13:40		FPM1-1-2	FPM1-2-2	FPM1-3-2	FPM1-4-2	FPM1-5-2	FPM1-6-2	FPM1-7-2	FPM1-8-2
13:40-14:00		FPM1-1-3	FPM1-2-3	FPM1-3-3	FPM1-4-3	FPM1-5-3	FPM1-6-3	FPM1-7-3	FPM1-8-3
14:00-14:20		FPM1-1-4	FPM1-2-4	FPM1-3-4	FPM1-4-4	FPM1-5-4	FPM1-6-4	-	FPM1-8-4
14:20-14:40		-	-	FPM1-3-5	FPM1-4-5	FPM1-5-5	FPM1-6-5	-	FPM1-8-5
14:40-15:00		Break							
	Session	FPM2-1	FPM2-2	FPM2-3	FPM2-4	FPM2-5	FPM2-6	FPM2-7	FPM2-8
		Analog Circuits and Filters III			Image Processing II	Electromagnetic Compatibility		Control Theory and Application II	Antenna Design and Analysis
	Chairperson	Dr. Banlue Srisuchinwong (SIIT)			Dr. Vuttipong Areekul (KU)	Assoc. Prof. Dr. Werachet Khan-Ngern (KMITL)		Dr. Mongkol Konghiran (KMUTT)	Asst. Prof. Dr. Chuwong Phongcharoenpanich (KMITL)
15:00-15:20		FPM2-1-1			FPM2-4-1	FPM2-5-1		FPM2-7-1	FPM2-8-1
15:20-15:40		FPM2-1-2			FPM2-4-2	FPM2-5-2		FPM2-7-2	FPM2-8-2
15:40-16:00		FPM2-1-3			-	FPM2-5-3		FPM2-7-3	FPM2-8-3
16:00-16:20		FPM2-1-4			-	FPM2-5-4		FPM2-7-4	-
16:20-16:40		-			-	FPM2-5-5		-	-
17:00		End of Conference							

Keynote Speech
Trend of HDD Research in Thailand

Brent Bargmann
Vice President of Operations,
Seagate Technology (Thailand) LTD.

Bargmann's career spans 20 years of hard disc drive engineering & manufacturing experience with the majority of his career spent in the Asia Pacific region. He joined Seagate in 1989 as part of the Imprimis/Control Data acquisition, where he started his industrial career. Bargmann assumed his current position as Vice President of Operations, Seagate Technology Thailand, in 2000. In this position, his responsibilities include general management & overall operations performance for all Seagate sites within Thailand. During his time in this position, Bargmann has been instrumental in leading the successful implementation & deployment of key manufacturing strategies and the expansion of the Korat manufacturing campus.

Other experience has included senior engineering & operational management leadership roles within Seagate Thailand. In 1997, he was appointed Vice President of Engineering for Seagate Thailand. In this role, he was responsible for all technical organizations within Thailand. He transitioned to Vice President of Tepaurk Operations in 1999, assuming the lead factory role for Seagate Thailand's largest manufacturing site, prior to assuming his current position.

Bargmann holds a Bachelor of Science degree in Electrical Engineering from South Dakota State University, Brookings, South Dakota.

He is the Chairman of IDEMA, Asia-Pacific Thailand advisory committee and an active member of IEEE & the National Society of Professional Engineers.

Building Agent-Based System Software Using Aspect-Oriented Framework[1]

Paniti Netinant

Computer Science Department
Bangkok University
Bangkok, Thailand
paniti.n@bu.ac.th

Abstract

Building intelligent concurrent systems that are able to adapt to environmental changes and reconfigure themselves is the key to avoiding performance degradation of concurrent real-time software systems and ensuring the liveness property of such systems. In this paper we present a machine learning-based approach that addresses the design of agent-based intelligent concurrent software systems in order to ensure the reliability and performance properties for such systems. Although reliability and performance are conflicting requirements in most cases, we will show how to use an aspect-oriented technology by which these requirements can be designed, implemented, reused, and replaced in isolation from each other. The performance and reliability of the software system can be reasoned about by intelligent agents who can direct the system to reconfigure itself in order to adapt to the environment changes. The agents rely on the data-mining techniques to discover patterns of performance degradation or imminent signals of reliability violation and to predict policies that cope best with the environmental fluctuations.

KEYWORDS: Aspect-Oriented Framework, Agents, Reusability, Extensibility, and System Software

1. Introduction

Concurrent real-time systems are designed mainly in order to ensure performance and the stringent reliability requirements, sometimes referred to as quality properties. Until recently, the design and development of these systems have inter-mixed the quality code with the functionality code for such systems; the quality properties cut across the functional components. This crosscutting phenomenon [7, 12, 13, and 14] generally breaks the component model, and makes it hard to design, and reuse, especially when the functional and non-functional requirements change. Generally requirement changes force reengineering for these systems. Generally, the functional components for these systems are stable. On the other hand, the quality requirements are volatile and reactive to the environment changes. Another issue that necessitates engineering reconfigurability when designing these systems is the make It possible to build these systems so that and real-time properties can be replaced while the systems are running, in order to adapt to environmental changes.

Building intelligent software systems that have open architectures, which support reconfigurability, is essential for concurrent real-time applications by which their well-being and performance are heavily dependent on their capability to cope with the environment fluctuations. The Mars pathfinder problem [19, 20] is a classical example of such systems where conflict of interest between performance and liveness properties has forced system reset that was due to a priority inversion problem. The Mars Pathfinder spacecraft's engineers were aware of the priority inheritance solution for such problems but they preferred not to use it [17], since it may cause performance degradation for the spacecraft. The lack of software adaptability hooks is the main reason that NASA system engineers chose not to deploy priority inheritance mechanism in order to avoid the system resetting.

Aspect-Oriented Programming (AOP) [1, 2, 7, 8, and 9] is a new programming paradigm that attempts to separate the functional components from the interaction components (aspects). Aspects are defined as properties that cut across groups of functional components. While these aspects can be thought about and analyzed relatively separately from the basic functionality, at the implementation level they must be combined together. Programming concerns manually into the system's functionality using current component-oriented languages results in aspects being tangle throughout the code. This code tangling makes the source code difficult to develop, understand and evolve because it destroys modularity and reduces software quality [8]. In this paper we show how to deploy aspect-oriented technology, which provides an architectural support for the design and development of intelligent concurrent systems. We show how the aspect code can be isolated from the functional components that otherwise would be intermingled with the code of the functional components. Isolating the functional components from the non-functional components, the aspect code, has many attractive benefits: first and foremost it promotes reusability for the functional classes and the aspect classes. It also simplifies

[1] This research has been supported by Thailand Research Funding (TRF) organization and Bangkok University. The contract is MRG4780168.

design of complex systems, since the interaction code separated from the functional code. Our approach is a step toward building reconfigurable intelligent systems and improves the software quality as it complements the object-oriented and component-oriented technologies with a set of design principles in order to engineer adaptability into software systems.

In our approach, assembling the intelligent concurrent systems from both functional and non-functional components through the use of aspect-oriented technology supports both static and dynamic adaptability when building the intelligent concurrent systems.

Recent advances in information technology have demonstrated that there are numerous issues that may affect the quality of service for software systems and that the only way to improve the quality of these services is by using software agents that monitor, advise, and react based on the environment changes. A software agent is an autonomous software component that can react to and interact with its environment. An agent is autonomous, since it runs in its own thread of control, and reactive, because of its capability to respond to incoming messages.

The agent-based approach is still in its infancy, although it is getting more popular as more IT applications are using this approach. Enterprise applications, B2B applications, Personal Agents, information filtering, Information monitoring, and interface agents / personal assistants just to name a few. For example, in information monitoring, activities are dependent on the timely notification of changes in the environment or in the data sources. Agents are very useful for monitoring different data sources for specific data. Agents can be dispatched to remote locations to monitor data sources. An interface agent is a program that able to operate within a user interface and actively assist the user in operating the interface and manipulating the underlying system. MS-Office assistants are an example of this category.

Aspect-Oriented Systems for Agents

Concurrent object-oriented open software systems are composed of functional requirements and concurrency requirements. Mixing the functional code and concurrency may impede code reuse; this has been documented in the literature as the inheritance anomaly problem. Solutions for the inheritance anomaly problem vary from domain specific languages (ABCL) to framework based solutions [4]. Framework solutions are preferred over domain specific languages, since these frameworks are based on common object-oriented languages and require less time to learn and deploy. Despite the success of the framework based approaches, support of static and dynamic adaptability for concurrent software systems has not been addressed in a formal way. Recent aspect-oriented approaches [4] have provided elegant solution to support the static adaptability aspect concurrent software systems, though the dynamic adaptability aspect has been left un-addressed. Aspect-oriented technology complements the object-oriented technology in order to avoid so-called code-tangling phenomena and support static adaptability for concurrent open software systems; Adaptability [5] is an important factor that enables software systems to evolve in order to meet future requirements. Reflective approaches [8, 9] offer solutions to support the dynamic adaptability aspect for open software systems, though reflection-based solutions have hard-coded decision processes that react and adapt to environment changes in an intelligent way.

Agent-oriented technology is getting more popular as more industrial applications start to deploy this approach. Agents are needed mainly to deal with uncertainty and react to environment changes and they are very useful to monitor systems resources and notify the interested parties to change their behavior to cope with the environment changes. Methods to engineer intelligence and machine learning within agents vary from data mining [17] techniques to q-learning [10]. In [17], the authors presented an approach based on the data-mining technique to discover patterns of behavior and network intrusion detection. And in [10, 11] the authors have demonstrated how q-routing algorithms can be used to discover best routes in a highly congested network.

Our approach integrates agent-oriented technology and aspect-oriented technology to build intelligent concurrent software systems [3, 6]; systems that run within an uncertain environment and require the capability of altering their components and policies during run time. Figure 1 shows our integrated view of these technologies. Our approach delivers a framework solution for building these systems. The agents are needed to assist in the decision making process for reconfiguring the software system during run time. Aspect-orientation techniques are used since they isolate the functional components from the aspectual components like performance and reliability.

Figure 1. A concurrent object as a cluster of components and aspects within the aspect moderator framework.

The aspect-oriented approach [4] has demonstrated its effectiveness in building concurrent object-oriented systems, where the concurrency aspects, like synchronization constraints and scheduling policies, are isolated from the functional components. This approach helps in building a stable software system that can easily adapt to meet future requirements and react to environment changes. The approach stated in [4] did resolve the static

ptability problem, but did not address the dynamic ptability aspect for building intelligent concurrent ftware systems. Our research has revealed the need to se intelligent components, agents that can aid the ncurrent software system in the decision-making ocess to reconfigure itself in order to adapt to nvironmental changes. The agents in our approach ploy the data mining technique and the Bayesian gorithm to monitor system resources and predicate terns of performance degradation or imminent signals reliability violation, and offer advice to react to these anges.

Aspect-Oriented Framework

A sequential object is comprised of functionality ntrol and shared data. Access to this shared data is ntrolled by synchronization and scheduling tractions. Synchronization controls enable or disable thod invocations for selection. The synchronization straction is composed of guards and post-actions. ring the Precondition phase, guards will validate the nchronization conditions. In the Notification phase, st-actions will update the synchronization variables. e scheduling abstraction allows the specification of heduling restrictions and terminate-actions. At the econdition phase, scheduling restrictions use scheduling unters to form the scheduling condition for each thod. At the Notification phase, terminate actions date the scheduling counters. During the Precondition se, the synchronization constraints of the invoked thod are evaluated. If the current synchronization ndition evaluates to resume the scheduling constraints e then evaluated. After executing the Precondition se, the moderator will activate the method in the quential object. During Notification, synchronization iables and scheduling counters are updated upon thod completion. The aspect moderator object ordinates functional and aspectual behavior, by ndling their interdependencies. We stress the fact that e activation order of the aspects is the most important rt in order to verify the semantics of the system. nchronization has to be verified before scheduling. A ssible reverse in the order of activation may violate the mantics. There are other issues that might also be volved. If authentication is introduced to a shared object r example, it must be handling before synchronization.

A major component of quality in software is liability: a system's ability to perform its job according the specification (correctness) and to handle abnormal uations (robustness). Under this theory, a software stem is viewed as a set of communicating components hose interaction is based on precisely defined ecifications of the mutual obligations known as ntracts. These contracts govern the interaction of the ement with the rest of the world. The importance of sertions is also stressed in [14] where it is described w the absence of specifications caused the disaster sociated with the European Ariane 5 launcher. The aspect moderator framework adopts this approach in a different context: defining assertions (preconditions and postconditions) as a set of design principles. Meyer argues that assertion monitoring yields to a productive approach to debugging, testing and quality assurance, in which the search for errors is not blind but based on consistency conditions provided by the developers themselves. As a result, reliability should be a built-in component in software development, not an afterthought. None of Java, Ada or CORBA has any built-in support for design by contract.

In [14] the authors argue that without specification it is probably safer to redo rather than to reuse. Another important issue is the one of the verification of components and aspects in isolation from each other. One must be able to test the functionality of a component as well as being able to test that an aspect will align nicely with the functional component. Otherwise, there can be no guarantee that components and aspects will co-operate. In other words, one must test and verify the collaboration of components and aspects. This would constitute an important phase in the design process.

In [11] the authors stressed the importance of aspect manifestation in every stage of development. The issue that in some cases aspects should remain run-time entities was also discussed in [5, 15] also stressed this issue by arguing that much like conditional compilation, aspects must be woven to the program on-demand. In technologies that rely on automatic weaving, aspects manifest in the model and in the program code, but neither in object code (byte code in the case of Java) nor in executable (binary) code. In [7] argues that with static weaving it might be impossible to adapting or replacing aspects dynamically. The framework manages to achieve the manifestation of aspects at run-time. We argue that is important that in order to achieve maximum flexibility a framework must provide for dynamic aspect evolution and ideally support both static and dynamic behavior. As an example, an aspect such as synchronization can be statically dealt with.

In ESP [8] and the Adaptive Arena [1] the functional part of a system is separated from the synchronization code, but it still remains in the same class. The separation of functional and aspectual code in the aspect moderator framework results in program code that is more modular. Furthermore, the framework follows a general-purpose approach in order to achieve composition of concerns. This way, it is not confined to certain aspects but can address a number of aspects. It is also language neutral. With the exception of AspectJ, current technologies are confined in domain specific languages. We introduce the concept of an aspect bank, where the moderator of a cluster initially needs to collect and register all the required aspects from. The aspect bank provides a hierarchical two-dimensional composition of the system in terms of aspects and components.

4. Conclusion

Agent-based software systems are the next wave in the software engineering discipline. Recent advances in

ware technology have stressed the need to build lligent open software systems in order to build mically reconfigurable software systems that can l with environment uncertainty and predict usage terns for system resources. A key factor for building ly reconfigurable software systems is to identify and te the aspectual code from the functional code in er to maximize reusability and facilitate nfigurability. Aspect-oriented programming is an rging programming technique that makes it possible gineer the reconfigurability of system software. We software agents in our framework in order to support decision-making processes for such systems. Our ework based on approach integrated aspect-oriented hnology and agent-orientation in order to support the mic adaptability aspect for intelligent concurrent ware systems, where agents can be asked to monitor em properties and react to undesirable events by nfiguring the software without halting the running em.

References

Bardou, D. (1998). Roles, Subjects, and Aspects. How Do They Relate? Position paper, *ECOOP '98* Workshop on Aspect-Oriented Programming , pp.55-59.

Berger L., Dery M. and Fornarino M. (1998). Interactions Between Objects: An Aspect of Object-Oriented Languages. Position paper, *ECOOP '98* Workshop on Aspect-Oriented Programming.

Callsen, C. J., and Agha, G. A. (1994). Open Heterogeneous Computing in Actor Space. *Journal of Parallel and Distributed Computing*, 1994, pp. 289--300.

Constantinides. C., Bader A., Elrad T. (1999)., A Framework to Address a Two-dimensional Composition of Concerns Position paper, *Object-Oriented Programming: Systems Languages and Applications (OOPSLA '99)* First Workshop on Multidimensional Separation of Concerns in Object-Oriented Systems. Denver, Colorado (USA). November 1, 1999.

Fayad, M., and Cline, M. (1996). Aspects of Software Adaptability, *Communications of the ACM*, 39(10), 1996, 58-59.

Kiczales, G., Lamping J., Mendhekar, A., Maeda, C, Lopes C., Loingtier,J-M; and Irwin J. (1997). Aspect-Oriented Programming. *In Proceedings of ECOOP '97*. LNCS 1241. Springer-Verlag, pp. 220-242.

Lopes, C. and Kiczales, G. (1998). Recent Developments in AspectJ. *Position paper ECOOP* 98 Workshop on Aspect-Oriented Programming.

Pryor, J.and Bastán, N. (1999). A Reflective Architecture for the Support of Aspect-Oriented Programming in Smalltalk. *Position paper, ECOOP'99* Workshop on Aspect-Oriented Programming.

[9] Bershad, B. Savage, S. Pardyak, P. Sirer, G. Fiuczynski M. Becker, D. Eggers, S. and Chamber, C.1999 Extensibility, Safety, and performance in the SPIN Operating System *ECOOP'98*.

[10] Boyan J. and Littman M.(1995). A Distributed Reinforcement Learning Scheme for Network Routing. *Technical Report CMU=CS-93-165*.

[11] Kiczales, G. Lamping, J., Mendhekar, A. Maeda, C Lopes, C. Loingtier, J-M., and Irwin J (1997 "Aspect-Oriented Programming". *PARC Technical Report,* SPL97-008P9710042 pp 69-71.

[12] Kiczales, G., Lamping, J., Mendhekar, A. Maeda, C Lopes, C., Loingtier,J-M., and Irwin, J.(1998). Aspect-Oriented Programming". In M.Askit and Matsuoka, editors, *Proceedings of the 12th European Conference on Object-Oriented Programming* ECCOP'98, Springer Verlag, Berlin pp 88-95.

[13] Kiczales, G. (1996), *Aspect-Oriented Programming Position Paper from the Xerox PARC Aspect-Oriented Programming Project,* Xerox PARC, Palo, Alto, CA.

[14] Kubat, M. (1998), "*A Machine Learning-Based Approach to Load Balancing in Computer Networks*, Cybermetics and Systems Journal, 23, 1992, pp. 389-400.

[15] Stolfo, S., Fan, W., Lee, W. Prodromidis, A., and Chan, P. *Proceedings of DARPA Information Survivability Conference and Exposition*, IEEE Computer Press, p. II 130-144, 2000.

[16] Boyan, J. A., and Littman, M. L. (1994). *Packet routing in dynamically changing networks: a reinforcement learning approach*. In Cowan, J. D. Tesauro, G., and Alspector, J., eds., Advances in Neural Information Processing Systems 6. Morgan Kaufmann Publishers.

[17] Boyan, J. A., and Littman, M. L. (1994). *Packet Routing in Dinamically Changing Networks: a reinforcement learning approach*, In Advances in Neural Information Processing Systems 6 (NIPS 6) 1994, 671—678.

[18] Diekmann, R., Frommer, A., and Monien, B. (1999 Efficient Schemes for Nearest Neighbor Load Balancing. *Technical report, Dept. Maths. Comp. Sc. Univ.* Paderborn, Furstenallee 11, D-33102 Paderborn Germany.

[19] Sha, L., Rajkumar, R., and Lehoczky, J.P.(199 Priority Inheritance Protocols: An Approach to Real Time Synchronization. *IEEE Transactions on Computers*, vol. 39, pp. 1175-1185.

[20] Wilner, D. (1997). " The Path Finder Invited Tal *The 18th IEEE Real-Time Systems Symposium*, San Francisco, December, 1997.