Applying Aspects within Modern Computing Domains

Saša Subotić, Judith Bishop

Department of Computer Science, University of Pretoria, South Africa

ABSTRACT

Aspect orientation is changing the way software is developed within various computing domains. In this paper we discuss the characteristics of aspect oriented programming, and the development of new or existing applications using aspect-oriented techniques and tools. We will show that this attractive technology is useful for solving the problems of code scattering and tangling within various computing domains. Aspects are emerging everywhere, and there is a particular need to introduce and practice them strategically in areas such as computer security, parallel programming and distributed computing. Our aspect research vehicle is the Algon distributed framework. The idea of Aspect Oriented Algon is proposed and investigated. The focus here is to show how aspect orientation can be applied to an Algon application aiming at a better separation of concerns. Basic examples are logging, debugging and performance related aspects associated with this system. We argue that this modern programming paradigm allows existing systems to be re-designed or modified as a viable approach to improving the functionality and flexibility of the system.

Computing Review Categories: C.2.4, C.4, D.1.3, K.3.2;

KEYWORDS: Paradigm, Aspect Oriented Programming (AOP), Aspect Oriented Software Development (AOSD), Algorithms on the Net (Algon), Aspect Oriented Algon (AOA), AspectJ, High Performance Computing (HPC), Distributed Computer Systems (DCS), Computer Security

1 INTRODUCTION

Computer scientists are fond of the word *paradigm*. A paradigm in the computing environment is a model used to describe the thinking about something or to show how something can be produced. During computer science education scientists are taught a particular paradigm. From that point on, scientists think and reason using that initial paradigm. It is difficult to get outside this paradigm for most scientists, which makes acceptance of new paradigms within the scientific community extremely difficult. Therefore, it is of utmost importance to perform a paradigm shift on a broader community of middle-aged computer scientists, in order to benefit from advantages of new knowledge and discoveries [18].

Object orientation has become one of the most popular programming paradigms. The OOP paradigm is based on the simulation of real world objects that comprise a system. The program consists of classes that describe objects that encapsulate all the state and behaviour of the associated real world entities. However, even with this approach, there is some functionality that could be encapsulated in an object and then scattered over a series of objects [9]. This problem of scattering and tangling created a shift in thinking from objects toward aspects. Aspects encapsulate a state and behaviour that is not an intrinsic part of a real world entity and form a central unit of the aspect oriented paradigm.

Aspect orientation is a new branch in this post-object programming era and is changing the way software is developed within various application domains. Four years ago it was officially established, by *MIT Technology Review*, as one of the top ten emerging areas of technology that will profoundly impact our economy and our lives [25]. Since then, most of the work on investigation of this paradigm has been done mainly in research laboratories around the world, but in South Africa its adoption proceeds slowly. There are no significant reporting results related to aspect orientation in South Africa. Particularly, there is a lack of presenting practical experiences with applying AOSD technologies.

Aspect oriented programming is not a replacement for object orientation. Simply stated, it is a collection of additional concepts that are added to OOP. Aspects present new challenges to software development practice, and we feel that there is a need to introduce them in different fields of computer science. Since there are many existing courses around the world addressing concurrency, distributed systems, computer security and high performance computing, we anticipate that one of the outcomes of this paper will be to stimulate thinking in terms of aspects in this direction.

The main goal throughout this paper is to provide the reader with an understanding of the aspect oriented programming and an appreciation for the different areas of research and study to which it can be

Email: Saša Subotić `sasa@tuks.co.za`, Judith Bishop `jbishop@cs.up.ac.za`

applied. The paper is divided into three parts, covering AOP foundation, its applications within various computing domains, and our current investigation.

Following this introductory part, section *2* describes the general idea behind the aspect oriented paradigm. This section further explains what kind of problems AOP can solve, the tools available, and lists the guidelines for effective use of AOP. Here we address both advantages and limitations of aspect orientation.

Section *3* describes the behaviour that emerge when aspects are used in computational, high performance, distributed systems and computer security fields. In order to illustrate how AOP applies to a specific computing domain, certain aspects are considered and applied to small-scale problem.

Section *4* explains how the Algon distributed framework can benefit by making use of this new approach. This section explores the Algon architecture, and investigates the idea of a two-level implementation of Aspect Oriented Algon.

Section *5* covers related work and presents the summary of achievements and ideas for further work.

2 ASPECT ORIENTATION: *FOUNDATION*

2.1 Aspect Oriented Programming

Aspect oriented programming is programming paradigm that specifically targets the management of crosscutting concerns. AOP enables developers to clearly separate crosscutting concerns that would otherwise be intertwined throughout an implementation. For example, a credit card processing system's core concern would process payments, while its crosscutting concerns would handle logging, security or performance.

As a simple illustration, consider Figure *1*, showing the difference between the OOP and AOP paradigm in managing crosscutting concerns. The OOP based approach creates a certain level of code scattering and tangling by forcing the core modules to embed the crosscutting concern's logic. On the other hand, with the AOP based approach you can make changes to an aspect or even replace it without affecting the other parts of the application.

AOP aims at improving the quality of the software by decreasing the level of code scattering and tangling. These two phenomena are also known as symptoms of non-modularization [15]:

- *Code scattering* is when a single issue is implemented in multiple modules.
- *Code tangling* is when a module is implemented to handle multiple concerns simultaneously (can not be solved with inheritance).

These two phenomena tend to appear together and they are primary symptoms of non-modularization. The implications of this non-modularization are poor traceability and quality, lower productivity and code reuse, and difficult evolution. All these features can be improved with AOP by decreasing the level of code scattering and tangling.

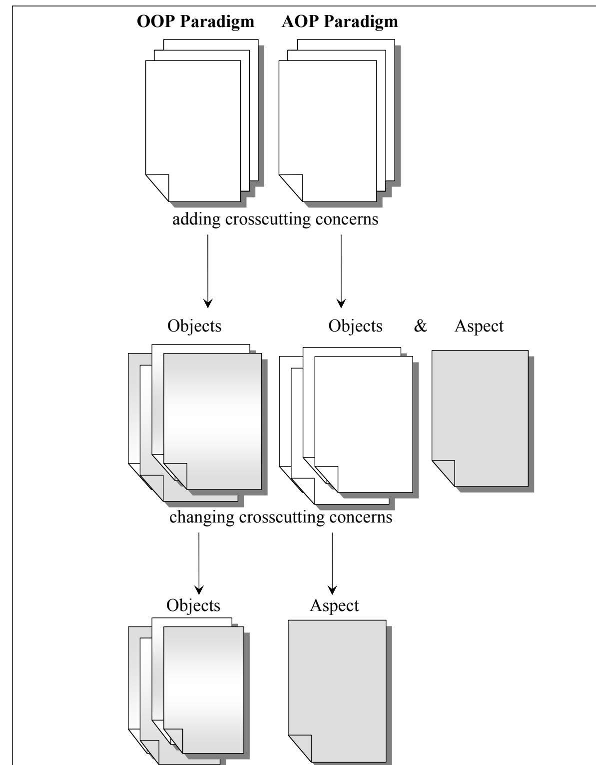

Figure 1: Management of crosscutting concerns with OOP and AOP

2.1.1 Core and Crosscutting Concerns

When dealing with AOP it is necessary to identify core and crosscutting concerns. A core concern captures the central functionality of a module, while crosscutting concerns capture system-level requirements that cross multiple modules. Logging, tracing, profiling, policy enforcement, pooling, caching, authentication, authorization and transactional management are some crosscutting concerns that can be nicely addressed with aspects. Table *1* identifies certain aspects for certain domains. From the given table it is clear that aspects could refer to location, communication, synchronization, etc.

While object orientation is the most common methodology employed today to manage core concerns, it is not sufficient for many above mentioned crosscutting concerns. Therefore a new AOP paradigm is introduced, which builds on success of object orientation, and specifically targets the management of crosscutting concerns. AOP utilize advan-

Domain:	*Distributed Comp.*	*Image Processing*
Aspects:	What the objects do	Op. on pixel maps
	Their location	Control structure
	Communication	Memory usage
	Synchronization	Sharing

Table 1: Example domains and key aspects of concerns in each, adapted from [13]

tages of OOP, where aspects observe object and inject code when required. The objects itself do not relate to the aspects thereby freeing the programmer from focusing on these crosscutting concerns.

2.1.2 Functional and Non-Functional Behaviour

There are mainly two ways of using aspects. One way is to separate concerns that cut across the functional component and another way of using aspects is to modify existing application in order to integrate a new feature.

Aspects are concerns that are dependent on the problem domain. A categorization can be beneficial towards the understanding the nature of aspects [6]. Aspects can address both functional and non-functional behaviour. Implementing a functional behaviour using aspects requires that at least certain structure for the function already exists in the system. On the other hand implementing a non-functional behaviour would have no impact on the design or implementation of the existing system.

Worrall et al [31] provide an example of the functional versus non-functional concern by considering a simulation of a CD-ROM drive. Two aspects, added to the CD-ROM drive, are the ability to write CDs and the ability to count the number of bytes read. The first one would be functional requirement of the new drive. On the other hand implementing a byte counter would be a non-functional requirement.

2.1.3 Development and Production Aspects

Categorizing aspects further, the AOP practitioners differentiate among two types of aspects [2]:

- *Development aspects*, which are used during the development phase, and then removed.
- *Production aspects*, which are used together with the final product.

Common development aspects are aspects for logging, tracing, debugging, profiling and testing. All these aspects can be used in the development process, and they can be removed or unplugged from the final application, as functionality is unaffected by these aspects. Furthermore, development aspect can be used as an ancillary part of system, where you can design the functionality to be deployed but disabled, and enable it when debugging.

Common production aspects are aspects for performance monitoring, updating and notification, error handling and security related functions. Production aspects are an essential part of the final system. When adopting the AOP incrementally, first start with development aspects (understand the system), and then use it to implement crosscutting concerns in production systems.

2.1.4 Pros and Cons of Aspect Orientation

AOP promotes clear design and reusability by enforcing the principle of separation of concerns. AOP explicitly promotes separation of concerns. Earlier paradigms did not do this so explicitly. This separation of concerns provides cleaner assignment of responsibilities, higher modularization and easier system evolution, and leads to a system that is easier to maintain and understand. Major benefits in using this attractive technology are improved time to market, more code reuse and reduced cost of feature implementation. This is achieved by collecting scattered concerns into a single structure unit (an aspect), which is designed in such a way as to promote understandability, maintainability, extensibility, reusability and adaptability.

Pros
The inventors think it is good
A growing user community think it is good
Many books regarding the topic
Many courses devoted to AOP
The AOP languages are evolving
There is a place for future development
Experiences show that AOP is stable
Future development can prohibit dangerous use
Cons
Many people still critical towards AOP
No quantitative results on benefits of AOP
No publications regarding the limitations
It looks simpler then it is
Pattern matching easy to break
Hard to see what happens in the source
AspectJ is not a mature technology
Too powerful ways to write bad code

Table 2: Pros and Cons of Aspect Orientation, derived from [10]

On the other hand AOP can not be applied in every situation. It is mostly suited for large scale developments. It is difficult to test and debug, and it is not yet proven to work under all conditions. It is often perceived as difficult to implement and hard to learn. Furthermore, the program flow in AOP based systems is hard to follow i.e. we are giving up the control of a detailed and understandable program flow. It is important to mention that it is still hard to find scientifically founded criticism in the literature regarding the topic. Only one example on AOP limitations is found by Kienzle J *et. al.*[14], suggesting that transaction management, through crosscutting, is hard to separate into an aspect. Finally, AOP breaks the encapsulation, but only in a systematic and controled way [15]. Table *2* further lists AOP pros and cons.

2.2 Aspect Oriented Communities

AOP is not yet ubiquitous in industry. However, it receives much attention from press and from companies such as IBM and Xerox. Particularly, IBM, BEA, and JBoss are supporting AOP in their products. IBM sees AOP as crucial technology for successful growth and they are investigating hugely in AspectJ (section *2.3.1*) and AspectJ Development Tools (section *2.3.2*). Microsoft is also interested in adopting AOP.

AOP generated its own conference. The conference is an annual event on Aspect Oriented Software Development, which promotes aspect orientation throughout the world. From here researchers can reach the Community Wiki where AOSD concepts, terms, tools, projects, events, and training courses are discussed. The next biggest community, supporting AOP, is European Network of Excellence on Aspect Oriented Software Development. Its mission is to promote AOSD activities in Europe and to strengthen innovation in applying AOP techniques. Asia is another continent with large community support. AOAsia is an Asian Network of research on Aspect Oriented Software Development, aimed at building AOP events and promoting AOP research.

AOP is receiving attention from many universities around the world, and particularly from universities in USA and Netherlands. There are many research labs and special interest groups within universities focusing on aspect oriented software engineering. Furthermore, an aspect oriented course has been proposed by Laufer et al.[17].

2.3 Aspect Oriented Languages

The purpose of aspect oriented language is to specify structured transformations on a program [27, 28]. This mainly involves inserting or removing the code at well defined points. In other words the main responsibilities of aspect oriented languages are to insert the code before points of interest, to insert the code after points of interest, or to replace the code at the points of interest. The most common locations for this insertions or removals are calls to functions, function definitions or pieces of functions [27].

Table *3* lists the current AOP languages. A comprehensive source of information about these languages is available from the AOSD community resources and libraries [1].

2.3.1 The AspectJ Programming Language

There are many implementations of AOP for many languages. AspectJ from Xerox PARC [2] is one such implementation of aspect oriented programming. It is built on top of the programming language Java and provides additional mechanisms to modularize crosscutting concerns. This aspect oriented extension to

Language	*Description*
AspectJ	AOP extension to Java, *section 2.3.1*
AspectWerkz	AOP framework for Java
JAsCp	AOP extension of Java
AspectSharp	AOP framework for .NET
Eos	AOP extension for C
AspectC++	AOP extension to C/C++
AspectS	AOP framework for Smalltalk
AspectXML	Aspect oriented approach to XML
AspectR	AOP extension for Ruby
AspectScheme	Scheme language supporting aspects

Table 3: Examples of AOP languages extracted from [1]

```
public aspect ExampleAspect {
  // Dynamic crosscutting
  pointcut doSomething()
  call ( * ExampleClass.do* ( .  .  )  );

  before() :  doSomething() {
  .  .  .  advice body
  }
  after() :  doSomething() {
  .  .  .  advice body
  }
  // Static crosscutting
  declare warning :  <pointcut>:<message>;
  declare error :  <pointcut> :  <message>;
  declare precedence :  Aspect1, Aspect2;
  declare parents :  [Child] extends [Class]
  private float ExampleClass._dataMemeber;
}
```

Figure 2: Two types of crosscutting in the AspectJ language

Java is the most widely adopted programming language supporting AOP.

In AspectJ applications, Java classes are used to implement core modularity and *aspects* (class-like constructs) are used to implement crosscutting modularity. In an AspectJ application everything revolves around *join points*. These are points in the execution of a program, where crosscutting concerns are woven in. According to AspectJ terminology there are two types of crosscutting (Figure *2*):

- *Static crosscutting* describes crosscuttings that influence the interfaces of the involved types and does not modify the execution behaviour of the system. AspectJ provides the following mechanisms to achieve this kind of influence :

  1. Introduction - introduces changes to the classes, aspects and interfaces of the system.
  2. Compile-time declaration - adds compile-time warnings and errors when certain usage patterns are captured.

- *Dynamic crosscutting* describes crosscuttings that influence the behaviour of an application. AspectJ provides the following language constructs to achieve this kind of influence :

  1. Pointcut - a constructor that selects join points and collects the context at those points based on different conditions.
  2. Advice - a method like construct, that executes before, after or around join points picked up by a pointcut.

With these additional constructs, the Java developer can add new functionality in the system without

changing any code in the core modules. AspectJ retains all the benefits of Java and therefore it is platform independent. It is used today for many real world projects. Its major practice is for enhancing the middleware platform, adding and improving security features to existing applications, and particularly for monitoring and improving performance [15]. Figure *4* (section *3.1.1*) provides a simple, yet powerful example of an AspectJ program.

2.3.2 Eclipse and AspectJ Development Tools (AJDT)

AspectJ project is run under the same open source as the Eclipse project and provides the most advanced AspectJ plug-in for an Eclipse IDE [4]. It is important that AspectJ version corresponds to the latest version of the AJDT, and supported version of Eclipse IDE.

AspectJ Development Tools provide tool support for editing, building and debugging AspectJ programs on the Eclipse platform. AJDT contains a lot of features designed to help you understand and be aware of the effects of advice in your program. The primary features are the editor and outline view [4].

Outline view shows all the places that the aspect advises. In other words, the contents of the outline view will contain, when an aspect or class is selected, new sections indicating where advice has been applied. The AJDT also provides another view of your project that graphically displays an overview of how your aspects are applied to your application. This is known as aspect visualizer, which shows visual overview of the effect of the aspects in the system. In addition to this visualization view there is a capability of generating documentation (similar to javadoc). A complete user guide for the AJDT plug-in can be accessed by visiting [2] and following the appropriate links.

The AspectJ technologies include a compiler (*acj*), a debugger (*ajdb*), a documentation generator (*ajdoc*), a program structure browser (*ajbrowser*) and support for IDE integration. Developers are encouraged to use AOP language with IDE support, as it tells them about advice and where it is added in the source code. Furthermore, IDE support can tell you when an aspect relies on a particular signature, making it less likely to delete something from a pointcut without noticing.

In Eclipse, it is as easy to run an AspectJ application as it is to run a traditional Java application.

Type	*Explanation*
upward compatible	all legal Java programs are legal AspectJ programs
platform compatible	all legal AspectJ programs run on standard Java VM
tool compatible	existing tools can be extended to work with AspectJ
programmer compa.	programming in AspectJ feels natural to Java programmers

Table 4: AspectJ as compatible extension to Java taken from [2]

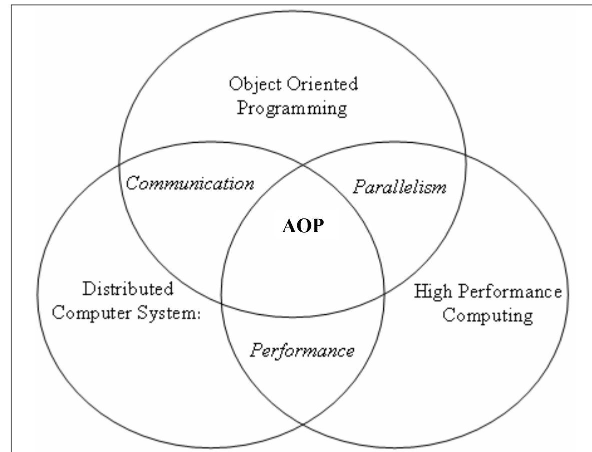

Figure 3: Emergent Behaviour of Aspects in HPC and DCS

This is because AspectJ is fully compatible with the Java programming language (Table *5*).

2.4 Emergent Behaviour of Aspects

Aspects are emerging in many areas of computer science. With aspects we can always influence the behaviour of the existing applications. We may change certain behaviour, or we may add additional ones, without even changing any code in the core modules of the existing application.

Emergence is defined as a process of complex pattern formation from simple rules [29]. This section illustrates emergence from chaos, as introduced by OOP in managing crosscutting concerns, to order by using AOP. Figure *3* shows the influence of object-oriented programming on distributed systems and high performance computing, and how AOP and aspects can build on it. We see that many entities operate in an environment, forming more complex behaviour. This emergence of complexity can be nicely addressed with aspects. The resultant emergent behaviour is not a behaviour of certain aspects, nor a behaviour of the core modules. It is the combination of the two that triggers it.

Emergent behaviour also occurs when the number of interactions between aspects in a system increases, thus potentially allowing for the creation of many new types of behaviour. However, a large number of interactions of multiple aspects can in fact work against the emergence of interesting behaviour. In the next section we provide simple illustrations on how to apply aspects and how to avoid such unexpected or unpredictable behaviour.

3 ASPECT ORIENTATION: *APPLICATION AND PRACTICE*

3.1 Aspects in High Performance Computing

Building a software system with high performance, high scalability and high security is a very big challenge. The main idea behind high performance systems is to use more then one computer or a computer

with more than one processor to solve a problem [30]. As a consequence, one can expect to achieve greater computational speed. Areas requiring this computational speed include mostly numerical and scientific applications. Traditional mechanisms for controlling parallel execution in numeric and scientific applications do not provide an easy separation of concerns. The problem of code tangling appears in many implementations of parallelised algorithms [12]. It is therefore important to investigate the use of AOP for parallel applications.

Simple examples of aspects here are tracing and performance analysis. In fact, for high performance computing applications, performance is an aspect that comes up very often. It is necessary to deal with both communication and computation separately and sum the components to form the overall parallel time. For example, on entering a computationally intensive method - start the clock, and on leaving this method - stop the clock, and calculate the difference and display overall time performance (see Figure *4*). A second concern is that high performance applications are not easy to understand and follow, due to their parallel nature, so developers often want to add some statements in the program to see what happens during execution. This kind of issue is a universal need in today's computing environment.

In principle it is possible to use AOP to separate core mathematical code from performance measuring code. Next subsection (Mandelbrot set application) covers the aspect oriented implementation of two concerns. One development concern (debugging-tracing) and the other a production concern (performance-time). These two aspects can not stand alone without the core concern, but they are not dependent on one another.

```
// a) OOP example of time measurement
public class TimeWatch {

   public TimeWatch() {
      difference = 0;
   }
   public void start() {
      startTime = System.currentTimeMillis();
   }
   public long stop() {
      endTime = System.currentTimeMillis();
      difference = endTime - startTime;
      return difference;
   }
   private long difference,startTime,endTime;
}

// b) AOP example of time measurement
public aspect TimeWatchAspect {

   pointcut measurePerformance()
      call ( * *.compute* ( .  .  )  );

   before() :  measurePerformance() {
      difference = 0;
      startTime = System.currentTimeMillis();
   }
   after() :  measurePerformance() {
      endTime = System.currentTimeMillis();
      difference = endTime - startTime;
      System.out.println (difference);
   }
   private long difference,startTime,endTime;
}
```

Figure 4: Implementation example of time concern for measuring performance.

3.1.1 Example: *Mandelbrot Set Application*

Table *1* in section *2.1.1* provides certain aspects for an image processing domain. Here, the focus is on the parallel Mandelbrot Set application - an HPC example of processing a bitmap image. The computation can be divided into a number of completely independent parts which are known as embarrassingly parallel and it is possible to spread the computation across several processors. Computation is embarrassingly parallel with respect to each pixel.

This kind of computing task has two primary concerns: the underlying mathematical model (complex iterating function) and the high performance execution (measuring the time) of the resulting computation. The first concern is the computationally intensive part of the program. The core concern of an application will exactly consist of implementing this functionality. This will be encapsulated inside the class *MandelbrotP.java*.

The second concern is the measurement of execution time, a feature that is usually wanted when dealing with intensive computing algorithms, in order to see the benefits of parallelising. This execution time measuring of the parallel Mandelbrot set should not interfere with the core component. Figure *4* illustrates an implementation of this concern with Java and AspectJ. With a Java-based approach such a concern would typically require the user to insert code between the lines of the main algorithm. Thereafter, if the user wants to remove this concern he/she needs to modify the main algorithm again. On the other hand, by using AspectJ and aspects it is possible to introduce this concern of performance measurement in the main application, and then remove it - if necessary, without even modifying the main algorithm. This aspect, *TimeWatchAspect.java*, will measure the time at different points in execution of the program. It activates 'startWatch' before advice and 'endWatch' after advice. The aspect is activated any time when the method name starting with the word 'compute' is called.

In addition to 'startWatch' and 'endWatch' we may want to incorporate some kind of a delay inside the application. This is illustrated in Figure *5*. The delay is implemented as advice and can be switched on and off as required.

This example has been tested in parallel, along with AspectJ, where Java RMI was used to illustrate

```
// a) OOP example of time delay
void delay(int millis) {

long start =System.currentTimeMillis();
long now = System.currentTimeMillis();

   while (now - start < millis) {
      now = System.currentTimeMillis();
   }
}

// b) AOP example of time delay
void around (int millis) :  pointcut
   performanceMeasurement(millis) {

   long start = System.currentTimeMillis();
   long now = System.currentTimeMillis();
   proceed()
   while (now - start < millis) {
      now = System.currentTimeMillis();
   }
}
```

Figure 5: Implementation example of delay feature

static task assignment. Results are provided in Table *5*. Another solution of interest is to try to investigate the possibility of adding MPI code within an aspect. A thorough investigation of aspects in HPC is provided in Harbulot [11, 12].

Sometimes, for educational purposes, it is necessary to trace the execution of a parallel program to see what each processor is doing, which one is delayed, and at which point of execution, etc. For that reason, the developer might add some extra statements for logging, debugging and tracing purposes. These can be addressed nicely with aspects. However, certain unexpected and unpredictable behaviour can emerge. It is possible to get poor performance results, due to interaction of these aspects with the performance aspect. The performance concern will not output valid performance results, because concerns like logging, tracing and debugging affect performance by introducing a non trivial input-output cost (see Table *5*). To achieve desired behaviour, for all high performance applications, it is important to remove all tracing and debugging code before collecting any kind of performance results. This can be achieved by disabling certain aspects from the final product, which is always possible, as AOP has the ability to weave in and out any aspects from the final application.

3.2 Aspects in Distributed Computer Systems

A distributed system is a collection of independent computers that appears to its users as a single coherent system [24]. The most important issues that we need to consider when dealing with such systems are the following: communication, processes, naming, synchronization, consistency and replication, fault tolerance, and security.

600x600 timing(plugged)	*Slave 1 (bottom)*	*Slave 2 (top)*	*Slave 3 (middle)*
logging (plugged)	106 820ms	107 000ms	189 969ms
logging (unplugged)	8 422ms	8 531ms	86 156ms

Table 5: Results from Mandelbrot application

Distributed computer systems are characterized by a substantial amount of complexity and therefore preservation of modularity is not possible for all concerns. In CORBA (industry-defined standard for DS) and RMI (remote method invocation) a number of problems have not been solved elegantly within such a distributed framework. In particular fault tolerance, fault detection, and eliminating the central point of failure are examples of concerns that are not easily addressed as these issues tend to span multiple components in distributed applications [17]. They may be addressed with aspects, but with caution, as serious shortcomings could emerge in handling of the failure semantics in a distributed environment [31].

Table *1* in section *2.1.1* provides certain aspects that are identified for the distributed domain where the AOP approach should be useful. In addition, the most noticeable examples of crosscutting concerns (aspects) for distributed computer applications are the following:

- *Synchronization* - implements all code for synchronization inside an aspect and distribute it to all existing classes that are responsible for supporting it.
- *Remote Access* - tends to cut across the implementation and could be factored out from all existing classes to an aspect.
- *Fault Tolerance* - when part of the system fails, a new behaviour emerges which uses certain aspects along with the rest of the system.
- *Security* - implements all code for security policy inside an aspect and distributes it to all existing classes that are responsible for enforcing that policy.

All of the above issues and particularly synchronization and fault tolerance tend to cut across the implementation, and they are primary source of complexity in distributed applications. AOP may present the potential to address these issues appropriately and to enforce a formal separation between the main functionality and these issues in distributed computer applications.

Security is one of the most difficult principles in distributed systems. Because of its importance the security related aspects are discussed in detailed in section *3.3*. The next subsection illustrates the example of failure handling in the distributed computer environment.

```
import java.rmi.RemoteException;

public aspect FailureHandling {
   final int MAX = 7;
   Object around() throws RemoteException
  :call(* *.get*(..)  throws RemoteException){
      int tryagain = 1;
      while (true) {
         try {
            return proceed();
         }
         catch (RemoteException e) {
            System.out.println("Found " +e);
            if (tryagain++ > MAX) {
               throw e;
            }
            System.out.println("\tReattempt");
         }
      }
   }
}
```

Figure 6: Implementation example of failure handling, adopted from [15]

3.2.1 Example: *Handling of Failure*

Aspect can implement the functionality to handle network failure. In a distributed environment this is very important task and maximum availability of the service must be achieved. If the service is down it is necessary to reattempt the operations. Aspect handles failure by reattempting the operation few times before giving up. Figure *7* illustrates the simple implementation of handling the failure by reattempting the original operations.

It is important to know how many times to reattempt the operation in order to achieve approximately 99% availability. This is relatively easy to test using AOP approach by modifying the given aspect. Depending on the failure rate (high or low rate) and using the simple mathematical formula we can estimate the number of attempts for maximum possible availability. Table *6* provides the analysis and results of 100 simulations for three different scenarios.

3.3 Aspects in Computer Security

Viega argues that developers will continue to write insecure code and to fix security problems only when someone happens to notice those problems [27]. There are two reasons for this:

- First reason is that developers are not very good (experienced) in writing secure software.
- Second reason is the difficulty in modeling security in the object oriented paradigm, because security is a concern that affects the system in a broad way.

The aspect-oriented approach should be employed to model security. In other words, the security information should be moved into separate and independent piece of code - aspect, instead of being part of the main program. With security independent aspects one can easily replace all insecure function calls with secure replacement calls. The main advantage of this approach is that developers can write only the main program, and allowing security specialist to focus on security properties. However, even if you have a reusable security aspect written by specialist, both developers and security experts need a sound understanding of the aspect orientation in order to apply it properly. The most common security related concerns are authentication, authorization, audit and encryption. Figure *7* illustrates the implementation of common security concerns.

Availability aspect(unplugged)	*25%*	*50%*	*75%*
No. of reattempts	15	7	4
aspect(plugged)	$1-0.75^{15}$	$1-0.5^{7}$	$1-0.25^{4}$
Availability approx.	98.66%	99.22%	99.61%
100 simulations	100%	100%	100%

Table 6: Results from failure handling

3.3.1 Example: *Authentication and Authorization*

Security is one of the most difficult principles in distributed systems. Security in such systems consists of many components including authentication, authorization, encryption and auditing. The first two components are closely related. Authentication verifies that you are a legitimate user, and authorization establishes access permissions for that user. In OOP

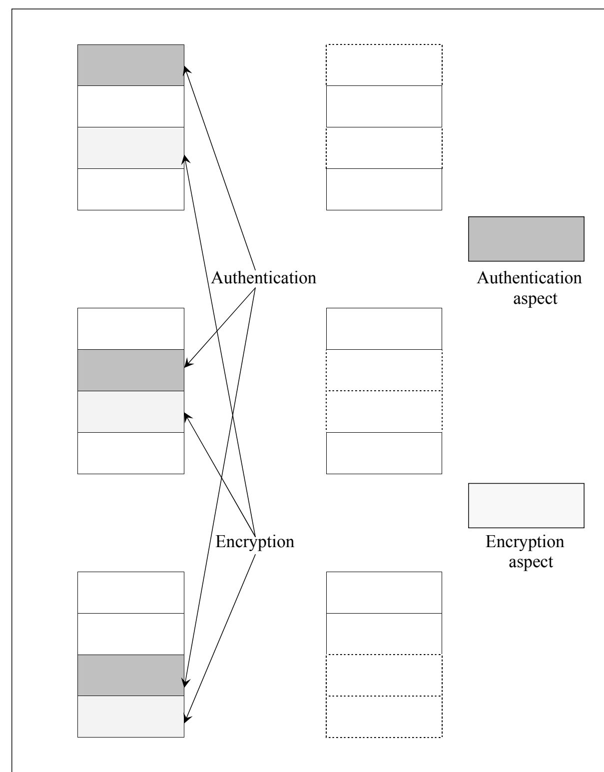

Figure 7: Security related crosscutting concerns (security aspects)

you will perform authentication in one module, pass the authenticated users to modules that need authorization, and then those modules will perform the authorization [15]. With AOP developers can create aspects addressing authentication and authorization and distribute them to all modules that need to perform these kinds of operations.

Authentication and authorization are so important in this highly connected computing world, and developers must be able to learn new ways of implementing such control. AspectJ based solution using the Java Authentication and Authorization Service (JAAS) is one of the newest ways to implement the above mechanisms in Java applications. Traditional methods such as JAAS APIs require you to modify multiple modules individually to provide them with authentication and authorization mechanisms. Now with aspect-oriented programming and AspectJ these mechanisms are not only easy to implement but also easy to evolve [15]. This implies that AspectJ works in cooperation with existing technologies, and not in competition.

However, a combination of these aspects and interaction with different modules can exhibit an emergent behaviour that is not desirable. The order of executing aspects, applied to the same point in execution, is unpredictable. Authorization can not be performed without first performing authentication, and therefore the authorization aspect must be executed after the authentication aspect. Also, it is not acceptable to disable the authorization aspect, as authentication alone is not sufficient. To enforce the desired behaviour it is necessary to consider rules and ways to control precedence of an aspect (Figure *8*). Unless you specify precedence, the order of execution is determined arbitrarily and may result in unexpected behaviour.

```
aspect AuthenticationAspect {

   pointcut AddToWrite():call(* *.write(..))
   before() :  AddToWrite() {
      // perform authentication
   }
}

aspect AuthorizationAspect {

   pointcut AddToWrite():call(* *.write(..))
   before() :  AddToWrite() {
      // perform authorization
   }
}

aspect SecurityPrecedenceAspect {

   declare precedence :
   AuthenticationAspect, AuthorizationAspect;
}
```

Figure 8: Authentication and authorization aspects

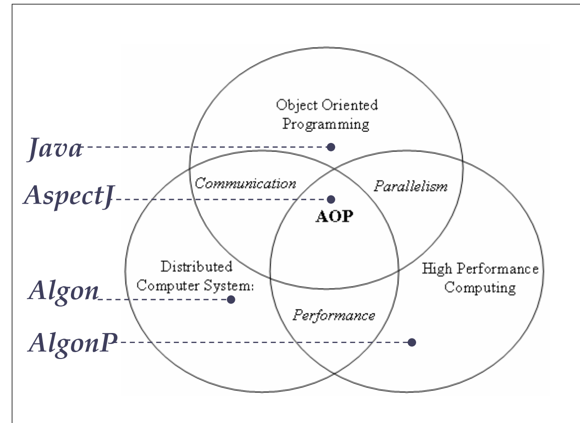

Figure 9: Algon and AlgonP applied to OOP/AOP model

4 CASE STUDY: *INVESTIGATION*

4.1 Algon and AlgonP

The research vehicle that drives our investigation of aspect orientation is the Algon system. It is a system begun in 2001 and it stands for ALGOrithms on the Net. Algon's main focus is to provide a framework for incorporating algorithmic software components into a distributed system. It hides complexity of distributed algorithms (for mutual exclusion, deadlock, etc) from programmers in a separate component layer. Now, it incorporates performance comparisons of the various distributed algorithms, and even further supports the interchangeability of the middleware on which it runs.

However, the design, and ultimately the implementation, of the system evolved in a rather ad hoc fashion. There is a need for better separation of concerns and the introduction of AOP. Algon is designed in such a way to allow for a clean separation of concerns. It provides library of algorithms that deal with many orthogonal, cross-cutting concerns that can be nicely plugged into the application logic. So why don't we use aspects with Algon?

As part of an industrially oriented project, with the Polelo Research Group, Algon is to be augmented with parallel algorithms. The new system will be known as parallel Algon or AlgonP. These algorithms require high speed computing and a framework for execution in parallel. Algorithms of this nature are also environment dependant and functionality is required to determine the best algorithm for a given environment. AlgonP is still under development. Figure *9* illustrates the model (section *2.4*) and idea that is used for investigation of aspects in Algon.

4.1.1 Overview of Algon Architecture

The Algon architecture consists of distributed algorithm and scheduling code grouped in one component layer. This component layer consists of at least one distributed algorithm (interface for a family of algorithms), one coded algorithm (implementing that interface) and one scheduler (for that family of algorithm). Figure *10* illustrates the Algon architecture.

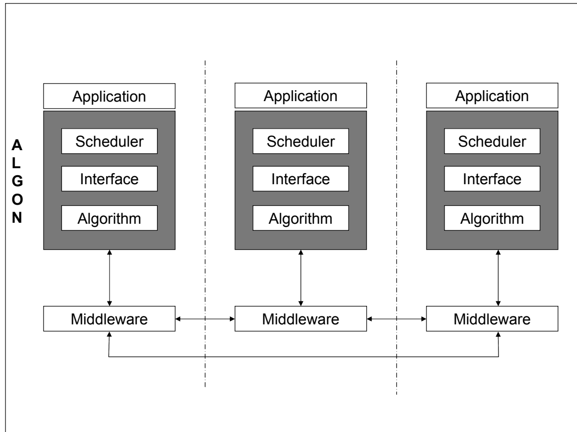

Figure 10: Algon architecture taken from [3, 22]

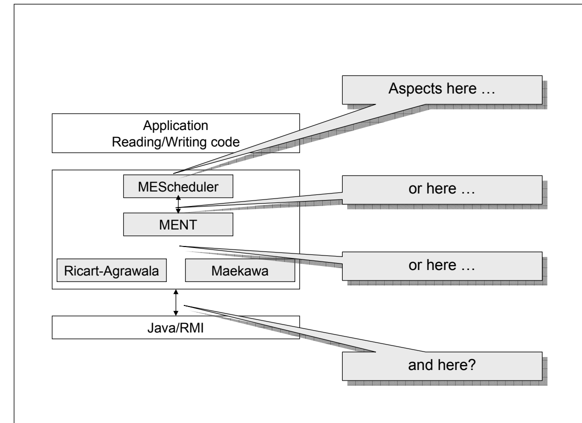

Figure 11: Aspect Oriented Algon - Algon and AOP

The scheduler is a class that provides transparent access to lower-level Algon features, and it performs configuration. It is specific to an algorithm family used and allows algorithm-dependent state information to be maintained outside of application logic. The interface abstracts behaviour common to functionally equivalent algorithms. Finally, the algorithm is programmed from theory and implements an interface. The greatest benefit of interfaces and schedulers is that minimal alteration is required to add one or more algorithms to an application. For more information about the Algon system and its architecture refer to the work of Bishop, Renaud and Worall [3, 21, 22].

4.2 Aspect Oriented Algon

Algon is entirely implemented in Java, due to Java's popularity and sustainability for the development of distributed applications, and provides classes which can be incorporated into new or existing application. Now, with AspectJ we can build on the existing Java application, as the language provides aspects which can be incorporated into such an application. Usage of the advanced features of the Java programming language, together with AspectJ constructs, will certainly contribute to the ideal separation of concerns.

Algon followed the approach of the traditional layered system architecture. The use of technology to isolate and encapsulate the concerns seems to make Aspect Oriented Programming the ideal alternative to a layered systems architecture. However, this aspectization has only limited application in the construction of distributed applications. Algon developers experienced that many kinds of specialised concerns that the Algon system addresses are neither orthogonal nor non-functional. They argue that Algon functional requirements are wrapped up in the functionality of the application and the extra dimension of error that must be dealt with simply can not be adequately expressed in an orthogonal way [31].

Programming the collaboration among the various processes running on different sites in a distributed application will often require the services of a distributed algorithm, and these algorithms need to be incorporated into the application code in some way. The conventional way would be to intersperse the algorithmic code with the application code and when required - injecting the implementation of the distributed algorithm at numerous points within the source code.

4.2.1 Implementation Proposal

It is important to note that there exist two distinct levels for aspect implementations in Algon:

1. The first level implementation is probably the reason for the current investigation - *aspects can be used to integrate the Algon into an application.*
2. The second level of implementation includes implementing Algon or parts of it using AOP - *thus separating out crosscutting concerns.*

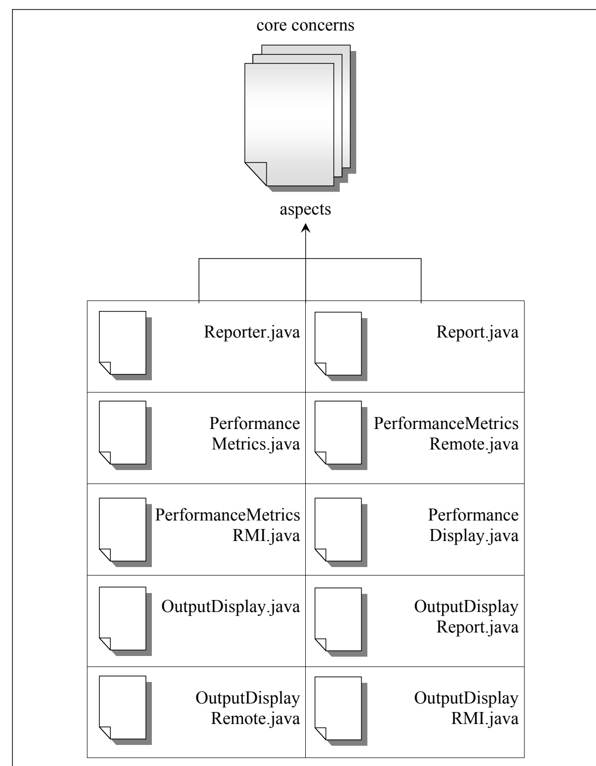

Figure 12: Performance concerns in the Algon system

In the former case, it should be possible for an application to access a distributed resource in which case Algon is integrated with the resource in such a way as to hide the distribute nature of the resource, as well as concurrency-related concerns. Such an application does not currently exist, and will only be part of the next phase of Algon's implementation. Distributed application will often require the services of a distributed algorithm, and these algorithms need to be incorporated into the application code in some way. A developer typically chooses one particular algorithm based on his/her understanding of what the system will need at design time. However, this is an unreasonable restriction due to dynamic and flexible nature of distributed systems. It would be far better to be able to replace the algorithm when the system has evolved or changed so that a different algorithm is required.

The Algon system provides this facility (to incorporate such algorithm into an application) with the minimum changes to the original code of an application. Now, we do not want to make these changes to original code. Instead, it would be better to create an aspect and inject the implementation of the distributed algorithm at numerous points within the source code (Figure *11*). This injection process is conceptually similar to the insertion of timing code or logging code into the program (section *3.1.1*).

In the second case, refer to Figure *12*. A number of Java classes forming Algon were identified with the aim of bringing out crosscutting concerns. The greatest and most obvious of these concerns were introduced by the implementation of performance-related aspects associated with the system.

```
public void getRequestSet() {

   UpdateQueue.uq.report(
      new OutputDisplayReport(
         algName, "Request set requested"));
      requestSet = this.configure();
   UpdateQueue.uq.report(
      new OutputDisplayReport(
         algName, "Configuration Complete"));
}
```

Figure 13: Code snippet illustrating performance-measuring code

```
public static boolean VERBOSE =
    (System.getProperty("verbose") != null);
                   .
                   .
                   .
         if (VERBOSE)
            System.out.println( .  .  .  );
         if (VERBOSE)
            System.out.println( .  .  .  );
```

Figure 14: Some debugging information scattered around the Algon system

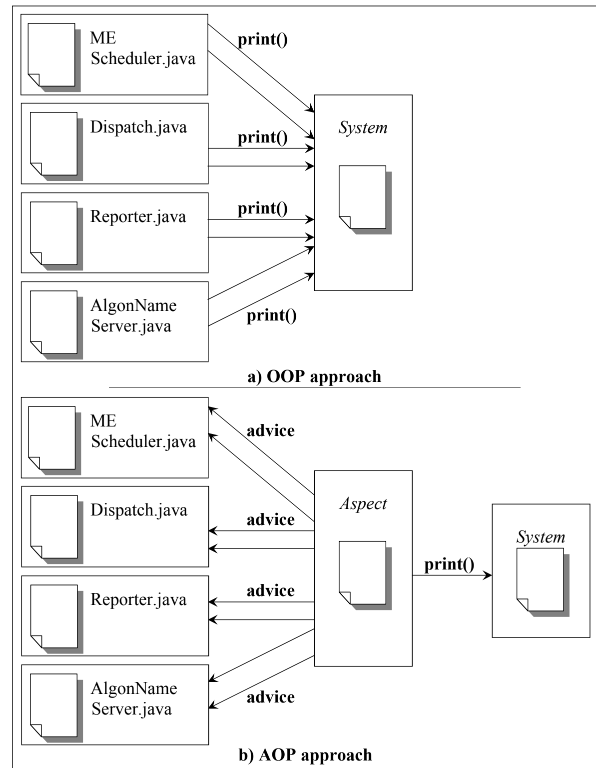

Figure 15: Difference between current and AOP approach in handling debugging information

In Figure *13* a snippet from *Algorithm.java* is taken, illustrating performance-measuring code. A few other cases also exist. In *MEScheduling.java* we found time delay implementation, which can be placed inside an aspect as already shown in Figure *5* in section *3.1.1*.

Figure *14* highlights a further highly re-occurring scenario. The statement 'System.out.println()' is scattered across different modular components. There are certain logging APIs which are used over the statement System.out.println (), but AOP promise to be a better approach. This is because with AspectJ we do not need to modify classes. All we need to do is to add an aspect to the system.

Figure *15* illustrates this behaviour in detail and provides an AOP approach. In traditional approaches the logging calls will be all over the core modules On the other hand, with an AspectJ-based approach the aspect separates the core modules and logger object, where the created aspect simply weaves in the logging invocations into the core modules when they are needed.

Following the above investigation, we conclude that the AOP approach has obvious and attractive advantages in dealing with many non-functional aspects of complex systems, but that it should be used with caution when dealing with functional aspects of system, especially with regard to distributed systems.

5 CONCLUSION

Although AOP does not solve any new problems, we proposed situations and approaches where AOP can

be useful for solving existing problems in a better and efficient way with much less time and effort. The main purpose was to stimulate thinking in terms of aspects along with objects and to promote wide acceptance of this new paradigm. Most importantly, by using AOP you will not only benefit from its advantages, but you will also become a better OO programmer.

We showed that distributed computer systems, computer security and high performance computing are promising fields where aspects could really make a difference. We illustrated how one can use AOP techniques and weave performance and security concerns in high performance and distributed applications.

The research work focused on the question whether and how AOP can improve distributed application development by looking at Algon distributed framework. Investigation of aspects in Algon has been done at a conceptual level, and in this paper we only touched on the appearance and use of aspects in Algon. Before doing any further work, it is important to explore in detail the limitations or weaknesses of aspect orientation in general and to show how these weaknesses would affect application development within high performance and distributed computing. Then we need to consider an approach that will incorporate AspectJ language while causing minimal destabilization of the overall Algon system.

We mentioned in this paper parallel Algon or AlgonP which is a latest advance for parallel algorithm exchange. Algorithms of this nature are also environment dependant and functionality is required to determine the best algorithm for a given environment. Therefore, we might extend our investigation of aspects from Algon to AlgonP.

There are plenty opportunities for AOP research. The following are example topics that can be further explored:

- Does it work for large multi-developer projects?
- To what kind of projects it is best suited for?
- How does one identify crosscuts and defines points?

Moreover, we think that particular focus of the further research should be on the software engineering issues, such as aspect oriented software development lifecycle (AOSDLC) and the emergence of aspect oriented teams (AOT).

Aspects are emerging in others research areas as well, such as aspects in concurrency, aspects in operating systems, aspects in databases and aspects for composition of web services. As a final note, computer scientists can expect to see many more changes (across various computing domains) over the next decade in programming techniques and programming languages in the direction that AOP started.

ACKNOWLEDGMENTS

Many thanks to John Muller, former member of the Polelo Research Group, for his initial work on Aspect Oriented Algon which formed a solid basis for AOP research. Many thanks to the South African Institute of Computer Scientists and Information Technologists for allowing us to present part of this work [23] at the SAICSIT 2005 annual international conference. The financial assistance of the National Research Foundation towards Algon research is hereby gratefully acknowledged.

REFERENCES

[1] "ASPECT ORIENTED SOFTWARE DEVELOPMENT". web site, www.aosd.net.

[2] "ASPECTJ TEAM". web site for AspectJ language, www.aspectj.org.

[3] BISHOP, J., RENAUD, K., AND WORRALL, B. "Composition of Distributed Software with Algon - Concepts and Possibilities". In *In Proceedings of Software Components*, Electronic Notes in Computer Science No: 65, pp. 1–12. Grenoble, April 2002.

[4] COLYER, A., CLEMENT, A., HARLEY, G., WEBSTER, M. *Eclipse AspectJ: Aspect Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools*. Addison Wesley Professional, 2005.

[5] *COMMUNICATIONS of the ACM*, vol. 44. Aspect Oriented Programming, www.acm.org/cacm, October 2001.

[6] CONSTANTINIDES, C.A, AND SKOTINIOTIS, T. "Reasoning About a Classification of Crosscutting Concerns in Object Oriented Systems". In *In second international Workshop on Aspect Oriented Software Environment of the SIG Object Oriented Software Development*. Bonn, February 2002.

[7] DRIVER, C., AND CLARKE, S. "Distributed Systems Development: Can we Enhance Evolution by using AspectJ?" In *Proceedings of the 9th International Conference on Object Oriented Information Systems*, vol. 2817, pp. 368–382. Springer: Lecture Notes in Computer Science, Geneva, Switzerland, September 2003.

[8] FRANCE, R., RAY, I., AND GHOSH, S. "Aspect Oriented Approach to early design modelling". In *IEEE Proceedings Proc.-Softw.*, vol. 151, pp. 153–185. August 2004.

[9] GABOR, L., AND MURPHY, J. "Using Aspect Oriented Programming for performance improving of J2EE applications". *Proc. of 6th International Conference on Technical Informatics, Transactions on Automatic Control and Computer Science*, vol. 49, pp. 223–226, Timisoara, Romania 2004.

[10] GIESE, M. "Introduction to Aspect Oriented Programming". Lecture notes, AOP course, Chalmers University of Technology, Sweden, www.cs.chalmers.se/˜giese/aop/course5.pdf, 2003.

[11] HARBULOT, B. *An Investigation of Aspect Oriented Programming*. Master's thesis, University of Manchester, England, 2002.

[12] HARBULOT, B., AND GURD, J.R. "Using AspectJ to Separate Concerns in Parallel Scientific Java Code". In *the Proceedings of the 3rd international conference on Aspect - Oriented Software Development (AOSD)*, pp. 122–131. March 2004.

[13] KICZALES, G., LAMPING, J., MENHDHEKAR, A., MAEDA, C., LOPES, C., LOINGTIER, J.M., AND IRWIN, J. "Aspect Oriented Programming". In I. M. Aksit and S. Matsuoka (editors), *ECOOP '97 - Object Oriented Programming 11th European Conference*. Finland, 1997.

[14] KIENZLE, J., AND GUERRAOUI, R. "AOP: Does it make sense? The case of concurrency and failures". In *Proceedings of ECOOP - Object Oriented Programming the 16th European Conference*, pp. 37–61. Jun 2002.

[15] LADDAD, R. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning, Greenwich, CT, 2003.

[16] LOPES, C.V., AND KICZALES, G. *D: A Language Framework for Distributed Programming*. Ph.D. thesis, College of Computer Science, Northeastern University, 1997.

[17] LAUFER, K., THIRUVATHUKAL, G.K., AND EL-RAD, T. "Enhancing the CS Curriculum with Aspect Oriented Software Development (AOSD)". Tech. rep., September 2003.

[18] ORR, K. "Managing Technology Decisionmaking". *Cutter consortium business IT strategies*, vol. 5, no. 9, pp. 1–29, 2002.

[19] PFLEEGER, C.P, AND PFLEEGER, S.L. *Security in Computing*. Prentice Hall, 3rd edn., 2003.

[20] PULVERMULLER, E., KLAEREN, H., AND SPECK, A. "Aspects in Distributed Environments". In *GCSE '99: Proceedings of the First International Symposium on Generative and Component-Based Software Engineering*, pp. 37–48. Springer-Verlag, London, UK, 1999. ISBN 3-540-41172-0.

[21] RENAUD, K., BISHOP, J., LO, J., VAN ZYL, P., AND WORRALL, B. "A Framework for Supporting Comparison of Distributed Algorithm Performance". In *Proceedings of 11th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2003)*, pp. 425–432. Genoa, Italy, February 2003.

[22] RENAUD, K., BISHOP, J., LO, J., AND WORRALL, B. "Algon: From Interchangeable distributed algorithms to interchangeable middleware". In *Proceedings of Software Composition*, pp. 59–78. Barcelona, Spain, April 2004.

[23] SUBOTIĆ, S., AND BISHOP, J. "Emergent Behaviour of Aspects in High Performance and Distributed Computing". In *Proceedings of SAICSIT 2005, South African Institute for Computer Scientists and Information Technologists*, pp. 11–19. White River, Mpumalanga, South Africa, September 2005.

[24] TANENBAUM, A.S., VAN STEEN, M. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Saddle River, NJ, 2002.

[25] *TECHNOLOGY REVIEW*. Emerging technologies that will change the world, www.technologyreview.com, January/February 2001.

[26] VAN ROY, P. "Aspect Oriented Programming for Distributed Systems: its use, its effect on language design, and its limits". In *Belgian Symposium and Contact Day on Aspect Oriented Programming and Software Evolution*. May 2004.

[27] VIEGA, J., BLOCH, J.T., AND CHANDRA, P. "Applying Aspect Oriented Programming to Security". *Cutter IT Journal*, vol. 14, pp. 31–39, February 2001.

[28] VIEGA, J., AND VEAS, J. "Can Aspect Oriented Programming Lead to More Reliable Software". *Quality Time, IEEE Software*, vol. 17, no. 6, pp. 19–21, 2000. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/52.895163.

[29] "WIKIPEDIA". the free online encyclopedia, en.wikipedia.org/wiki.

[30] WILKINSON, B., AND ALLEN, M. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Pearson Prentice Hall, Upper Saddle River, NJ, 2005.

[31] WORRALL, B., RENAUD, K., AND BISHOP, J. "Limitations of the Aspect-Oriented Approach in Distributed System Architectures". Tech. rep., Polelo Research Lab, July 2004.

ABOUT POLELO RESEARCH GROUP

The *Polelo* Research Group, under Prof Judith Bishop, covers distributed systems, high-performance computing, language understanding, web-based computing and software engineering. It is a part of the Department of Computer Science at the University of Pretoria. This specific project, on the Investigation and Practice of Aspect Orientation, is part of research being undertaken at the Honours level.