"To my taste the main characteristic of intelligent thinking is that one is willing and able to study in depth an aspect of one's subject matter in isolation, for the sake of its own consistency, all the time knowing that one is occupying oneself with only one of the aspects. ...

… The other aspects have to wait their turn, because our heads are so small that we cannot deal with them simultaneously without getting confused. This is what I mean by 'focussing one's attention upon a certain aspect'; it does not mean completely ignoring the other ones, but temporarily forgetting them to the extent that they are irrelevant for the current topic. ...

… Such separation, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts that I know of. ...

… I usually refer to it as 'separation of concerns', because one tries to deal with the difficulties, the obligations, the desires, and the constraints one by one. ...

… When this can be achieved successfully, we have more or less partitioned the reasoning that had to be done — and this partitioning may find its reflection in the resulting partitioning of the program into 'modules' — but I would like to point out that this partitioning of the reasoning to be done is only the result, and not the purpose. ...

… The purpose of thinking is to reduce the detailed reasoning needed to a doable amount, and a separation of concerns is the way we hope to achieve this reduction. …
The crucial choice is, of course, what aspects to study ‘in isolation’, how to disentangle the original amorphous knot of obligations, constraints and goals into a set of ‘concerns’ that admit a reasonably effective separation. ...

- Dijkstra, A discipline of programming, 1976
  last chapter, In retrospect

... To arrive at a successful separation of concerns for a new, difficult problem area will nearly always take a long time of hard work; it seems unrealistic to expect otherwise. ...

- Dijkstra, A discipline of programming, 1976
  last chapter, In retrospect

... The knowledge of the goal of ‘separation of concerns’ is a useful one: we are at least beginning to understand what we are aiming at.”

- Dijkstra, A discipline of programming, 1976
  last chapter, In retrospect

goal of this talk

- discuss the implementation of complex software systems
- focusing on issues of modularity
- how existing tools help achieve it
- propose a new tool to help improve modularity in some cases where existing tools are inadequate

slides, papers and system at www.parc.xerox.com/aop...

format of this talk

- sharing context
- a problem and an idea
- our current instantiation of the idea
- implementation
- summary and hopes

I sharing context
the engineering challenge

- extremely complex systems
- more than our mind can handle all at once
- must manage the complexity

problem decomposition

- break the problem into sub-problems
- address those relatively independently

solution construction

& composition

- construct complete systems from the designs by
  - implementing the sub-parts, and
  - composing them to get the whole

design & implementation

- decomposition breaks big problems into smaller ones
- composition builds big solutions out of smaller ones

“clean separation of concerns”

we want:
  - natural decomposition
  - concerns to be localized
  - explicit handling of design decisions
  - in both design and implementation

achieving this requires...

- synergy among
  - problem structure and
  - design concepts and
  - language mechanisms

  “natural design”

  “the program looks like the design”
the “component”\textsuperscript{1} concept

- a modular unit of functionality
- fits many natural design concerns
- well-supported by existing programming technology
- a rich collection of
  - design principles, conventions and notations
  - programming mechanisms

object-orientation

“objects”
- used in design and implementation
- object-oriented design
- object-oriented programming
- many tools to bridge the gap

procedural programming

“procedures”
- used in design and implementation
- some tools even bridge the gap

summary so far

good separation of concerns in both design and implementation

II

a problem and an idea

a distributed digital library
the component structure

- use objects
  - objects are a natural fit for this system
- so...
  - the design breaks down into component objects
  - implement using OOP

the class graph

<table>
<thead>
<tr>
<th>Book</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>holds</td>
<td>cooperates</td>
</tr>
<tr>
<td><strong>title</strong>: string</td>
<td><strong>author</strong>: string</td>
</tr>
<tr>
<td><strong>isbn</strong>: int</td>
<td><strong>pdf</strong>: pdf</td>
</tr>
<tr>
<td><strong>tiff</strong>: tiff</td>
<td></td>
</tr>
</tbody>
</table>

User

accesses

Printer

public class Printer {
  String status = "Idle";
  Vector jobs;

  public boolean print (User u, Book b) {
    PostScript ps=get_ps(b);
    Job newJob = new Job (ps, u.get_name());
    return queue(newJob);
  }

  boolean queue(Job j) {
    //...
    return true;
  }
}

class Library  {
  Hashtable books;
  Library() {
    books = new Hashtable(100);
  }

  public Book getBook(User u, String title) {
    System.out.println("REQUEST TO GET BOOK "+title);
    if(books.containsKey(title)) {
      Book b = (Book)books.get(title);
      System.out.println("getBook: Found it: "+b);
      if (b != null) {
        if (b.get_borrower() == null) {
          b.set_borrower(u);
          return b;
        }
      }
    }
    return null;
  }
}

class User {
  private String name;
  Library theLibrary;
  Printer thePrinter;
  public User(String n) { name = n; }

  public boolean getBook (String title) {
    Book aBook = theLibrary.getBook(this, title);
    thePrinter.print(this,aBook);
    return true;
  }
}

class Book {
  private String   title;
  private String   author;
  private String   isbn;
  private PostScript  ps;
  private User   borrower;
  public Book(String t, String a, String i, PostScript p) {
    title = t;
    author = a;
    isbn = i;
    ps = p;
  }

  public User get_borrower() {return borrower;}
  public void set_borrower(User u) {borrower = u;}

  public PostScript get_ps() { return ps; }
}

the code

all is well

- design is natural
- code looks like the design
- good separation of concerns
  - localized in the design
  - localized in the code
  - handled explicitly

a distributed digital library

minimizing network load

dataflow patterns
minimizing network load
controlling slot copying

method invocations
title, author, isbn
printable format

why?
• why did so much code change?
• why wasn’t this concern well localized?
• why didn’t this “fit” the component structure?

because...
• we are working with “emergent entities”, and
• the component concept, and its associated implementation mechanisms, fundamentally don’t provide adequate support for working with emergent entities

emergent entities
• emerge\(^1\) during program execution
  – from (possibly non-local) interactions of the components
• are not components
  – do not exist explicitly in the component model or code

\(^1\)emerge: to become manifest; to rise from or as if from an enveloping fluid; come out into view
emergent entities

- emerge\(^2\) during program execution
  - from (possibly non-local) interactions of the components
- are not components
  - do not exist explicitly in the component model or code

\(^1\) emerge: to become manifest: to rise from or as if from an enveloping fluid; come out into view

are tough to handle because...

- they are not explicit in the component model or code
- they have non-localized origins and interactions
- they cross-cut the component structure...

cross-cutting the components

- the sub-parts of the objects are not existing classes
- the desired dataflows are not existing message sends

but, but, but... the code can be remodularized to “fit” better...

violates separation of concerns leads to tangled code

claim

- remodularizing isn’t good enough!
  - it ruins the separation of concerns
- the functionality and the network optimization concern are fundamentally different
- would like different “carvings”\(^1\) of the system
  - in terms of component structure,
  - and in terms of emergent entities,
  - with support for the cross-cutting modularities

\(^1\) carve: to cut with care or precision, to cut into pieces or slices, to work as a sculptor or engraver

just try it

dataflow Book {Library to User}
{copy: title, author, isbn};

dataflow Book {Library to Printer}
{direct: pdf, tiff};
what it says

dataflow Book {Library to User}
{copy: title, author, isbn};

the dataflow of books, from library objects to user objects, should be implemented by copying the title, author and isbn slots only

how it says it

dataflow Book {Library to User}
{copy: title, author, isbn};

each book in the library is related to the user via the loan relationship

and...

dataflow Book {Library to Printer}
{direct: pdf, tiff};

print the book's title, author and isbn;

assume a...

- a “language processor” that
  - accepts two kinds of code as input;
  - produces “woven” output code, or
  - directly implements the computation

general claim

- remodularizing the component structure is not a satisfactory way of dealing with emergent entities
- want different carvings of the system:
  - keep the clean component structure
  - control emergent entities in “natural terms”
    - in terms of the emergent entity
    - with support for cross-cutting
emergent entities

- an entity that does not exist explicitly in the component model or code, but rather arises during execution
  - data flows
    - all the places this value goes...
  - control states
    - two methods running concurrently
    - one method blocked on another
    - all the callers of this function
    - history of calls up to this point (aka the stack)...

the “aspect” concept

- components are modular units of functionality
- aspects are modular units of control over emergent entities
- in the distributed digital library:
  - library component
  - book component
  - user component
  - printer component
  - …

“aspect languages”

- aspect languages
  - connect to a component language, and provide:
    - a mechanism for referring to emergent entities
    - a mechanism for exercising some control over the implementation of the emergent entities
    - support for using cross-cutting modularities

dataflow Book {Library to Printer} {direct: pdf, tiff};

summary so far

AspectJ is…

- an extension to Java™
- targeted at distributed and/or concurrent applications
- several general-purpose aspect languages
  - remote data transfer aspect language
  - computation migration aspect language
  - coordination aspect language
- a weaver for those languages
a data transfer aspect language

- provides control over data transfers between execution spaces
  - transfer of arguments and/or return values
  - control over sub-fields, sub-sub-fields etc.

referring to the emergent entity

```java
portal Library {
    Book find (String title) {
        return: Book: {copy title, author, isbn;}
    }
}
```

copy transfer mode

```java
data transfer aspect language

- provides control over data transfers between execution spaces
  - transfer of arguments and/or return values
  - control over sub-fields, sub-sub-fields etc.

referring to the emergent entity

```java
portal Library {
    Book find (String title) {
        return: Book: {copy title, author, isbn;}
    }
}
```
aspect composition cross-cuts too

- these aspects compose along dataflows
- not along normal class/method composition

more on cross-cutting

what this is and isn’t

- weaver combines two kinds of code
- equivalent effect of complex tangled code
- equivalent elegance of original clean code
  - component code is unchanged
  - natural modularity of aspects

a coordination aspect language

fits object-oriented modularity

- per-object
- per-class

cross-cuts object-oriented modularity

- per-object
- per-class
- multi-class
- any methods
status of AspectJ

- some preliminary user studies complete
  - results promising, not yet conclusive
- first public release to go on web-site shortly
  - free use (including in products)
  - weaver, documentation, example programs
  - coordination aspect language only
- next release early June
  - remote data transfer aspect language
- later releases
  - other aspect languages, operate directly on class files…

for example

```
public class Shape {
  protected int w = 0;
  protected int h = 0;
  int getWidth() {
    return w;
  }
  int getHeight() {
    return h;
  }
  void adjustLocation() {
    x = longCalculation1();
    y = longCalculation2();
  }
}
```

```
class Library {
  Hashtable books = new Hashtable(100);
  public Book find(User u, String title) {
    frob();
    if (books.containsKey(title)) {
      frob();
      Book b = (Book)books.get(title);
      if (b != null) {
        frob();
        if (b.getBorrower() == null) {
          frob();
          b.setBorrower(u);
        }
      }
    }
    return null;
  }
}
```

“frob every method call”

```
class Library {
  Hashtable books = new Hashtable(100);
  }
  public Book find(User u, String title) {
    frob();
    if (books.containsKey(title)) {
      frob();
      Book b = (Book)books.get(title);
      if (b != null) {
        frob();
        if (b.getBorrower() == null) {
          frob();
          b.setBorrower(u);
        }
      }
    }
    return null;
  }
}
```

domain transforms

- what is diffuse in one domain is local in another
- the Fourier transform moves between the two
  - it localizes what was non-local and vice-versa

what aspect weavers do

- implement one or more aspect languages
- allow us to program in alternate modularity
  - in the modularity of the emergent entity
  - help with cross-cutting
- aspect weaver must “gather up the roots and contact points of emergent entities”
  - places spread around the OO program
  - this can appear difficult...

“frob every method call”

```
class Library {
  Hashtable books = new Hashtable(100);
  }
  public Book find(User u, String title) {
    frob();
    if (books.containsKey(title)) {
      frob();
      Book b = (Book)books.get(title);
      if (b != null) {
        frob();
        if (b.getBorrower() == null) {
          frob();
          b.setBorrower(u);
        }
      }
    }
    return null;
  }
}
```

for example

```
public class Shape {
  protected int w = 0;
  protected int h = 0;
  int getWidth() {
    return w;
  }
  int getHeight() {
    return h;
  }
  void adjustLocation() {
    x = longCalculation1();
    y = longCalculation2();
  }
}
```

```
for example

```
\textbf{coordinator} Shape, Screen { \\
\textbf{mutex} \{adjustLocation, getX\}; \\
\textbf{mutex} \{adjustLocation, getY\}; \\
}
```

```
\textbf{public} \textbf{class} Shape { \\
  \textbf{protected} int w = 0; \\
  \textbf{protected} int h = 0; \\
  \textbf{int} \text{getWidth}(); \{ \text{return} \text{w}; \} \\
  \textbf{int} \text{getHeight}(); \{ \text{return} \text{h}; \} \\
  \textbf{void} \text{adjustLocation}(); \{ \text{\textbf{frob}();} \\
  \textbf{frob}(); \} \\
  \textbf{void} \text{adjustLocation}(); \{ \text{\textbf{frob}();} \\
  \text{\textbf{frob}();} \} \\
}
```

```
\textbf{class} Library { \\
  \textbf{Hashtable} books = \text{\textbf{new}} \textbf{Hashtable}(100); \\
  \}
  \text{\textbf{public}} \text{Book} \text{find} (\textbf{User} u, \textbf{String} title) \{ \\
    \textbf{frob}(); \\
    \textbf{if} (\textbf{books}.\text{\textbf{containsKey}}(\text{title})) \{ \\
      \textbf{frob}(); \\
      \text{Book} b = (\textbf{Book})\textbf{books}.\textbf{get}(\text{title}); \\
      \textbf{if} (b \neq \textbf{null}) \{ \\
        \textbf{frob}(); \\
        \textbf{if} (b.\text{\textbf{getBorrower}()} \neq \textbf{null}) \\
          \textbf{frob}(); \\
        b.\text{\textbf{setBorrower}}(u); \\
      \}
    \}
    \textbf{return} \textbf{null}; \\
  \}
}
```

IV
implementing aspect weavers

jump to conclusion
reflection links two domains
• the object domain: localizes books and their functionality
• the meta domain: localizes "frob every method call"

class Library {
  Hashtable books;
  Library() {
    books = new Hashtable(100);
  }
  public Book find(User u, String title) {
    if(books.containsKey(title)) {
      Book b = (Book)books.get(title);
      if (b != null)
        if (b.get_borrower() == null)
          b.set_borrower(u);
      return b;
    }
    return null;
  }
}

aspect weavers

can require a variety of domain-transforms
• method calls (all, per-class, per-selector…), field accesses (…), methods (…);
• who else is running
• where will this value go next
• •

summary

improved separation of concerns in both design and implementation
• complex systems
• design practices support decomposition into components & aspects
• programming languages support implementation and composition of components & aspects
  aspect-oriented design
  aspect-oriented programming

an analogy

(what I hope aspects are like)

different kinds of picture

- simple statics
- more detailed statics
- simple dynamics

conclusions

programming languages support implementation and composition of components & aspects

www.parc.xerox.com/aop

complex systems

improved separation of concerns in both design and implementation

design practices support decomposition into components & aspects

programming languages support implementation and composition of components & aspects

aspect-oriented design

aspect-oriented programming

Summary

Conclusions

An analogy

Different kinds of picture

Reflection links two domains

Aspect weavers
a distributed digital library

different kinds of picture

modeling of functionality
• modeling of control over emergent entities

different kinds of program

• programming with different carvings of the system
• allows clean separation of:
  – programming of functionality
  – programming of control over emergent entities

objects & aspects

• AOP enables modular control over emergent entities
• using languages that support cross-cutting modularities