

Università degli Studi di Pavia  
Facoltà di Ingegneria  
Corso di Laurea in Ingegneria Elettronica

Reducing the Effects of Growth Saturation on Web  
Sites Production with the Adoption of a Publishing  
Framework Based on XML Technologies

Il Candidato:

Stefano Mazzocchi

Il Relatore:

Prof.re Ivo De Lotto

Il Corelatore:

Dott. Alberto Biancardi

Anno Accademico 2000–2001

A Pissi e Papi:  
il vostro coraggio sarà la mia forza.  
Sempre.

# Preface

The work with this thesis was carried out independently and within the Robotic Lab of the Department of Computer Science at the University of Pavia.

First, I would like to thank my advisors, Dott. Alberto Biancardi and Prof. Ivo De Lotto, for their guidance and patience during my studies.

Then, I would like to thank my entire family, Laura C., Laura P. and all my friends for their incredible support during the long journey of my studies.

Sincere thanks go to the Apache Software Foundation for supporting my work on publishing frameworks and special thanks go to the Apache Cocoon worldwide community without which many of my ideas would have remained much more academic than they are today.

Many thanks go to Dott. Gianugo Rabellino for betting on my research concepts and providing precious feedback and suggestions along with useful case studies.

Finally, special thanks go to the Pinnock Family without whom my English would have never been good enough to participate to worldwide communities and acquire such wonderful learning experiences.

Stefano Mazzocchi

Pavia, June 2001

# Abstract

This thesis analyzes the effects of saturation trends that appear during production of web sites as their size grows and the number of people involved increases. A simple mathematical model that much such environment is proposed and used to identify the issues.

Next, the use of software design methodologies applied to work organization backed up by a web publishing framework based on XML technologies is presented as a solution.

Finally, we describe the implementation of such framework and identify a possible organization layout, indicating what technologies can be used to enforce it in order to reduce the impact of growth saturation and increase productivity and work quality in web site production.

# About The Author

**S**tefano Mazzocchi was born in Pavia, Italy, January 20, 1975. As an exchange student, he graduated from the Siuslaw High School of Florence, Oregon, U.S.A in 1992. Returned to Italy, he graduated from the Liceo Scientifico N. Copernico of Pavia in 1993. Driven by curiosity of the inner working of computers and electronic devices, the same year he enters the faculty of electronic engineering of the University of Pavia, still cultivating his hobby for computer programming.

In 1997, he gets his first contact with worldwide software development communities in the Apache JServ Project becoming user, then developer, than reaching release coordinator status by election (the higher role for that community) for the release of the 1.0 version.

In 1998, along with other community key players, he starts turning the focus from servlet engines to a more general web-oriented development based on the use of the Java language. In that period, he writes the Apache JMeter testing tool and he bootstraps the JAMES (Java Apache Mail Enterprise Server) Project and the Avalon Project for the creation of a server framework that would enforce the use of modern software engineering practices and concepts on open developed software.

For this work on servlet engines, he is invited by Sun Microsystems to join the Servlet API Expert Group responsible for the design and maintenance of the Servlet API for the Java Platform. He'll remain in this group until early 2001, helping with the creation of the 2.1, 2.2 and 2.3 versions of the specification.

For his work on the various Java projects under Apache, he is proposed for membership and accepted as a member of the Apache Software Foundation in 1999.

During that period, he shifts his focus on XML technologies and creates the the Apache Cocoon project and contributes to the Apache Ant project. In the same year, he helps the creation of the Apache XML Project, helping to bootstrap the Apache FOP community first and the Apache Batik community the year later.

For his involvement in XML technologies, he is invited to join the JAXP (Java API for XML) Expert Group where he'll remain until early 2001, helping with the creation of the 1.0 and 1.1 versions of the specification.

# Table of Contents

<b>Preface</b> .....	<b>iii</b>
<b>Abstract</b> .....	<b>iv</b>
<b>About The Author</b> .....	<b>v</b>
<b>Table of Contents</b> .....	<b>vi</b>
<b>List of Figures</b> .....	<b>ix</b>
<b>Introduction</b> .....	<b>1</b>
Digital Publishing .....	1
Networks as Costless Support .....	2
The Need for a Distributed Information System .....	2
HTML Glues the Pieces Together .....	3
Today's Web .....	4
<b>The problem of Growth Saturation</b> .....	<b>5</b>
Growth Saturation .....	5
Understanding the Reasons Behind Growth Saturation .....	9
Environment with Message Broadcasting and High Overlap .....	10
Environment with Message Broadcasting and Limited Overlap .....	10
Environment with Direct Messaging and No Overlap .....	11
The Economical Aspect .....	12
Conclusions on Growth Saturation .....	13
<b>An Optimization Strategy</b> .....	<b>14</b>
Partitioning by Concern .....	15
Concern Overlap and Mismanagement Bottlenecks .....	15
Framing Concerns .....	16
<b>Framework Design</b> .....	<b>18</b>
The Flat Concern Layout .....	18
Content .....	18
Style .....	19
Logic .....	19
Crosscutting Concerns .....	19
Design and Maintenance of the URI Space .....	20
Content Aggregation .....	20
Flow Design .....	20
An Improved Concern Layout .....	20
A single marketing–production channel .....	21
Limitations of This Approach .....	22
Difficult reshaping of existing structures .....	22
Necessity of wider skill diversity .....	22
The Necessity of a Technological Foundation .....	22
<b>Enabling Technologies</b> .....	<b>23</b>
Software Platform .....	23
Web Serving Environment .....	24
Data Description Languages .....	24
The Limits of HTML .....	24

A Step Back Two Steps Forward .....	25
The eXtensible Markup Language .....	25
Transformation Languages .....	27
Presentation Languages .....	29
The Babel Problem .....	30
<b>The Apache Cocoon Project .....</b>	<b>31</b>
Main Architectural Concepts .....	33
Event Based Processing .....	34
Centralized Resource Management .....	35
Matchers and Selectors .....	35
Readers .....	36
Actions .....	36
<b>Contract Analysis .....</b>	<b>37</b>
The Required Contracts .....	37
Design–Content .....	37
Design–Style .....	38
Design–Logic .....	38
Content–Style .....	39
Content–Logic .....	39
Style–Logic .....	39
Possible Critiques to This Analysis .....	39
Resulting Layout .....	40
<b>Case Studies .....</b>	<b>41</b>
Operaweb .....	41
The problem .....	41
Separation of Concerns .....	42
Structure .....	42
Case study 1: a “legacy” web site .....	42
Case study 2: a “cross media” web site built using XML technologies .....	43
Borland JBuilder 5 .....	45
Other Live Sites .....	45
<b>Conclusions .....</b>	<b>46</b>
<b>Future Work .....</b>	<b>47</b>
Semantic Web .....	47
Extending The Study on Web Applications .....	48
Content Management Systems .....	48
<b>Appendix A .....</b>	<b>50</b>
Separation of Concerns (SoC) .....	50
Inversion of Control (IoC) .....	51
Object–Oriented Frameworks .....	51
<b>Appendix B .....</b>	<b>53</b>
<b>Appendix C .....</b>	<b>56</b>
Good URIs Don’t change .....	56
Student’s Home Page .....	56
Dynamic Document Server .....	57
Bad URIs Do Change .....	58
Designing a Good URI Addressing Scheme .....	58

**Appendix D** ..... **60**  
**Bibliography** ..... **68**

# List of Figures

Figure 1 – Simple News Page.....	6
Figure 2 – Fancier News Page.....	8
Figure 3 – Growth Saturation.....	9
Figure 4 – Total Team Direct Time.....	10
Figure 5 – Direct Time per Individual.....	10
Figure 6 – Direct Time per Individual.....	11
Figure 7 – Total Team Direct Time.....	11
Figure 8 – Direct Time per Individual.....	12
Figure 9 – Total Team Direct Time.....	12
Figure 10 – The Flat Concern Layout.....	18
Figure 11 – Hierarchical Concern Layout.....	21
Figure 12 – java.apache.org Main Page.....	32
Figure 13 – Cocoon Pipeline.....	33
Figure 14 – Resulting Hierarchical Concern Layout .....	40
Figure 15 – OperaWeb Front Page.....	44
Figure 16 – Cocoon integrated into JBuilder 5.....	45

# Introduction

## What is web publishing?

The Webster English Dictionary defines *publishing* as:

To make public; to make known to mankind, or to people in general; to divulge, as a private transaction; to promulgate or proclaim, as a law or an edict.

For many historians, the invention of writing has been one of the most important inventions for mankind. It separates history from pre-history. This because it allowed knowledge to be exchanged, and information to survive its creator.

This is a very important concept: oral communication is synchronous since it requires both the speaker and the listener to be present when the communication takes place. Using writing, the information is stored and can be used asynchronously, even if the writer is not around, or, more important, is not even alive.

In oral communication, the concept of publishing was not present: the act of speaking automatically involved the act of publishing information to all the listeners. With writing, this changes radically: the act of writing and the act of publishing are well separated and generate the need for different technologies.

In fact, the second most important invention, the press with mobile characters, does almost nothing for writing, but it radically changes the concept of publishing. In fact, for the first time in history, the act of publishing becomes industrialized.

This invention created another revolution: not only information can be stored and retrieved asynchronously, but can also be copied with a fraction of the effort required. Not only the press amplified the signal, but it allowed costs to be reduced and information to be much more widely available to entire parts of the population that previously didn't have that chance.

### **Digital Publishing**

With the advent of computers and digital technology, information could be stored in ordered sequences of bits and it was not only limited to text, but also included sound, images, video, genetic sequences, computer programs, etc.: everything that can be serialized into sequences of bits becomes digital information and can be copied and published in the exact same way.

This fact greatly simplifies the concept of publishing since the same technology that is used to publish and distribute one type of information (in this case "digital") can be used to publish every information this broad type can convey. It creates a common denominator that can be used to transmit all sorts of information to all sorts of media and even mixing them ("multimedia").

The other great advantage of digital information is the relatively high resistance to copy degradation: digital copies are guaranteed to be bit-by-bit identical to the original, thus removing the original concept of a *copy* which implicitly meant signal degradation.

But the most significant impact of digital information comes from the reduced marginal costs of

copying the information. In fact, making a digital copy is a matter of creating another ordered stream of bits using the original as instructions for the copier, which must simply turn the state of the created bits depending on the original stream.

While it is true that digital publishing has the advantage of having reduced marginal costs for copies, it is generally false that this, alone, allows these costs to tend to zero, even for great volumes.

The problem is that as long as digital information is distributed on hardware supports (optical disks, magnetic disks, magnetic tapes, etc.), these all have fixed costs that can never be removed.

### ***Networks as Costless Support***

One of the solutions to remove the need for storing local copies of the information is having direct access to the location where this information is permanently stored. The client/server paradigm and the existence of digital networks allowed information to be kept in a single location and copied on request, streamed over the requesting client and consumed by the client with no fixed costs associated with the copy.

The fixed costs were now associated with the existence of the network and the maintenance of the information but are now independent (at least on a first approximation) on the number of copies requested.

For the first time in history, information can be published with marginal costs per copy which tend to zero very rapidly and present costs only on higher orders (for example, with bandwidth, memory or computing power limitations).

While digital publishing cannot be considered by itself any different from analog publishing (in fact, the record industry didn't change much from selling analog LPs to digital CDs) apart from the quality of the copy and the reduced marginal costs, it is along with digital network and direct streamed transport that digital publishing unleashes its full power.

### ***The Need for a Distributed Information System***

It was the '80s when the scientific community reached the technological ability to connect all computer networks in the world (what is now known as *the internet*) and allow digital information to be published at no marginal cost per copy, but soon faced the problems associated with this new publishing medium.

The challenge faced was new: all existing information systems were highly centralized while the internet protocols showed that a stable and scalable solution for network systems required heavy distribution and lack of single points of failure or growth bottlenecks.

Following these principles, in the early '90s, a team of computer scientists lead by Tim Berners-Lee (at that time at CERN in Geneva, Switzerland) created the what is now known as the *World Wide Web (WWW)* or simply the *web*.

Berners-Lee and his colleagues started from the concept that in order to allow distribution to take place, such information system should be flexible yet focused, easy to implement and easy to deploy, both on the server side and on the client side.

The other important concept was uniform resource identification: each source of digital information became, for the web architecture, a *resource* with a unique identifier (Uniform Resource Identifier [URI]) and each resource, given the identifier, could be retrieved using a stateless (thus easier to implement) protocol named HTTP (HyperText Transfer Protocol [HTTP/1.1]).

These two technologies (URIs and HTTP) represent the foundation of the web because they provide a unique and fully scalable addressing scheme along with a way to access the indicated information.

Given this technology, publishing something on the web means to create a resource that wraps that information and to make known its identifier.

While URI and HTTP provided the ability to publish any digital information and make it accessible from the entire internet, alone they would have made a very poor technology from a usability point of view: they would have been able, in fact, to create nothing but a list of unconnected resources, making the browsing experience very poor.

The third key concept in the web architecture is *hyperlinking*: the ability to connect a resource, or parts of it, with another. This is the reason for the definition of this information system: the *world-wide web* is a uniformly addressable collection of digital resources which can be linked one another and retrieved indistinguishably from all over the world.

The three concepts together create a web of information which can be *browsed* by everyone who has access to the internet. The ability to hyperlink resources creates the concept of a *web site*: a collection of resources which are stored in the same host.

The URI scheme, in fact, is composed by three main parts (plus other additional details which don't matter in this introduction):

```
protocol://host/resource
```

where

- **protocol** indicates the protocol used to obtain the resource (for example, "http"). In short, indicates to the retrieving agent (the browser or similar) *how* the resource should be requested.
- **host** indicates the unique identifier of the machine which hosts the resource. This identifier can be an internet IP address (which is a number) or a host name identifier, which can be looked up by special internet systems (DNS) and translated in its IP address.
- **resource** identifies the resource on that host.

The combination of the three elements allows URI schemes to be totally scalable because each host name is guaranteed to be unique by the internet naming authorities which regulate both the IP addresses and the domain names.

It is then sufficient to regulate the resource mapping scheme within the site to create a persistent and one to one relationship between a particular digital resource and its uniform identifier.

## ***HTML Glues the Pieces Together***

The original intent of the *world-wide web* was to create a simple way for scientists all over the world to publish their information and make it available to the rest of the scientific community.

In order to make the system scalable, the responsibility of updating and maintaining the information should have been distributed as well, optimally, placed directly on the information owner's shoulders.

In order to do this, the web architects understood the need for a simple textual language that allowed scientists to write and publish their information directly and have access to the hyperlinking capabilities that the web architecture gave them.

This language was created with the name *HyperText Markup Language* [HTML] and was born with

the intent of allowing non-technical people to author their own information resources (also referred to as *pages*, even if this is somewhat semantically incorrect) without the need for special tools.

To meet the proposed goals, the format should have been textual, and not binary, to allow any text editor to edit the resource. Moreover, the syntax should have been friendly and as intuitive as possible, making it really simple for everyone to grasp the ideas behind the design without a full understanding of network technologies or other technical details.

The choice went to the syntax already defined by an ISO standard called Standard Generalized Markup Language [SGML] which was already used extensively in centralized information systems. HTML started as a simple language to describe how information should be displayed on the user screen, along with hyperlink information.

Here is an example of a simple HTML file:

```
<html>
  <head>
    <title>This is an HTML page</title>
  </head>
  <body>
    <p>This is a paragraph</p>
    <center>
      <p>This is a centered paragraph</p>
    </center>
    <p>This is a <b>bold</b> word</p>
    <p><a href="http://mysite.com/">This is an hyperlink</a></p>
  </body>
</html>
```

The syntax is very easy to grasp even without reading any manual or technical documentation and a few tries are sufficient for people to start modifying existing pages or create new ones.

## Today's Web

Today, the World-Wide Web is the biggest and most used informative system ever known to mankind. Millions of individuals use it for many different things that range from finding information to shopping, from checking their bank accounts to chat with remote friends.

The web architecture has proven itself to be incredibly successful in allowing individuals and organizations to publish and make available their content on the web by reducing their marginal costs virtually to zero.

This allowed the number of resources available on the web to grow exponentially with time.

At the same time, the unforeseen success of the web in environments which it was not designed for, shows design limitations and second order effects that might limit its further evolution.

# The problem of Growth Saturation

## Expanding the Original Web Model

The web was designed to allow individuals or small groups of individuals to publish and exchange their information at very little total costs and virtually no marginal cost per published copy. The public information on the web was freely available and reflected the spirit of the scientific community which is based on information sharing rather than information selling.

But as more people got connected to the internet, the web started to appear as a virgin commercial landscape and ventures started to try business models to make profit out of web publishing, attracted by the virtually eliminated distribution costs.

The most used way to make a profit out of web site followed the TV business model: advertising. Web sites started to provide services to attract people and advertise themselves as *information portals* that allowed them to obtain whatever information they wanted simply going through their site gates.

Then, they sold themselves as *advertising space* to other companies.

This trend was driven by commercial needs but also increased the overall usability of the web allowing more and more non-technical people to find what they needed on the web and helped turning the web into the hub where all information can be retrieved and exchanged.

The original web model was designed around the notion of a highly distributed information system where each information resource was managed by a single person or a small group of people. HTML was designed exactly with that purpose in mind and it has proven itself great in those cases where the information follows this distribution.

On the other hand, portal-like sites created *information clusters* by placing continuously more resources into the same site in order to attract more people and show higher hit rates to increase their commercial value.

### **Growth Saturation**

HTML was designed for small sites and made simple to allow non-technical people to be able to manage and author web pages directly.

For this reason, in the original HTML model there is no notion of those separate skills that are required to author a web page: all these skills were needed, but expected to be found in the same person that does the authoring job.

But as soon as more information is added and the complexity of the site grows, one person is not able to manage it alone, thus more people is needed to author, update and manage that information.

Normally, these people have different skills and search different types of satisfaction in their work, but since most existing web technologies were designed around the original HTML model, thus with individuals in mind, these differences are not reflected in the design of the technology that drives web publishing.

The original HTML model was designed to scale globally, but wasn't designed to scale *locally*,

where for local scalability we mean the ability of several individuals with different skills to collaborate for the creation of the published information.

As early as the first commercial web sites were born, it appeared evident that three major different skills were required: those regarding the technological details that allowed web publishing to take place, those regarding textual content editing and those regarding visual content creation (such as layout, style, etc.).

Since the HTML model wasn't designed around this concept, no architectural help was originally designed to allow these groups to work independently from one another. Rather the opposite: in many circumstances, they were *forced* to work on the very same resource, with all the problems associated with concurrent access on the same information.

The first obstacle was found between content editors and graphic designers which shared the same *canvas* to draw their information, but their contributions were intermixed in the same page.

As an example of this, the following is a page that contains virtually no style information but only textual content:

```
<html>
  <head>
    <title>Some News page</title>
  </head>
  <body>
    <h1>News</h1>
    <p>
      <a href="http://www.some.site.com/">
        <strong>This is the first news!</strong></a> -
      <strong>10 Oct 2010</strong> -
      Blah blah blah blah blah blah blah blah blah blah blah blah
      blah blah blah blah blah blah blah blah blah blah blah blah
      blah blah blah blah blah blah blah blah blah blah blah blah
      blah blah blah blah blah blah blah blah blah blah blah blah
      blah.
    </p>
    <p>
      <a href="http://www.another.site.com">
        <strong>Another incredible news!</strong></a> -
      <strong>13 Feb 2120</strong> -
      Blah blah blah blah blah blah blah blah blah blah blah blah
      blah blah blah blah blah blah blah blah blah blah blah blah
      blah blah blah blah blah blah blah blah blah blah blah blah
      blah.
    </p>
  </body>
</html>
```

and here is how is rendered by a web browser:



Figure 1 – Simple News Page

and this is another page which is equivalent from a textual content point of view but much more complex:

```
<html>
<head>
  <title>Some News page</title>
</head>
<body bgcolor="#ffffff">
  <center>
    <table cellpadding="0" cellspacing="0" bgcolor="#000000"
      width="100%" border="0">
      <tr>
        <td>
          <table cellpadding="5" cellspacing="2" width="100%"
            border="0">
            <tr>
              <td bgcolor="#F0F0F0">
                <table cellpadding="3" cellspacing="0" width="100%"
                  border="0">
                  <tr>
                    <td align="center" width="100%">
                      <table cellspacing="10" width="100%" border="0">
                        <tr>
                          <td width="100%" valign="top">
                            <table cellpadding="0" cellspacing="0"
                              bgcolor="#000000" width="100%" border="0">
                              <tr>
                                <td width="100%">
                                  <table cellpadding="4" width="100%" border="0">
```

```

<tr>
  <td align="right" bgcolor="#C0C0C0">
    <big><big><b>News</b></big></big>
  </td>
</tr>
<tr>
  <td align="left" bgcolor="#E0E0E0">
    <a href="http://www.some.site.com/">
      <strong>This is the first news!</strong></a>
    </td>
</tr>
<tr>
  <td align="left" bgcolor="#ffffff">
    <strong>10 Oct 2010</strong> -
    Blah blah blah blah blah blah blah blah blah blah blah blah
    blah blah blah blah blah blah blah blah blah blah blah blah
    blah blah blah blah blah blah blah blah blah blah blah blah
    blah blah.
  </td>
</tr>
<tr>
  <td align="left" bgcolor="#E0E0E0">
    <a href="http://www.another.site.com">
      <strong>Another incredible news!</strong></a>
    </td>
</tr>
<tr>
  <td align="left" bgcolor="#ffffff">
    <strong>13 Feb 2120</strong> -
    Blah blah blah blah blah blah blah blah blah blah blah blah
    blah blah blah blah blah blah blah blah blah blah blah blah
    blah blah blah blah blah blah blah blah blah blah.
  </td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>
</body>
</html>

```

which is rendered on a web browser as

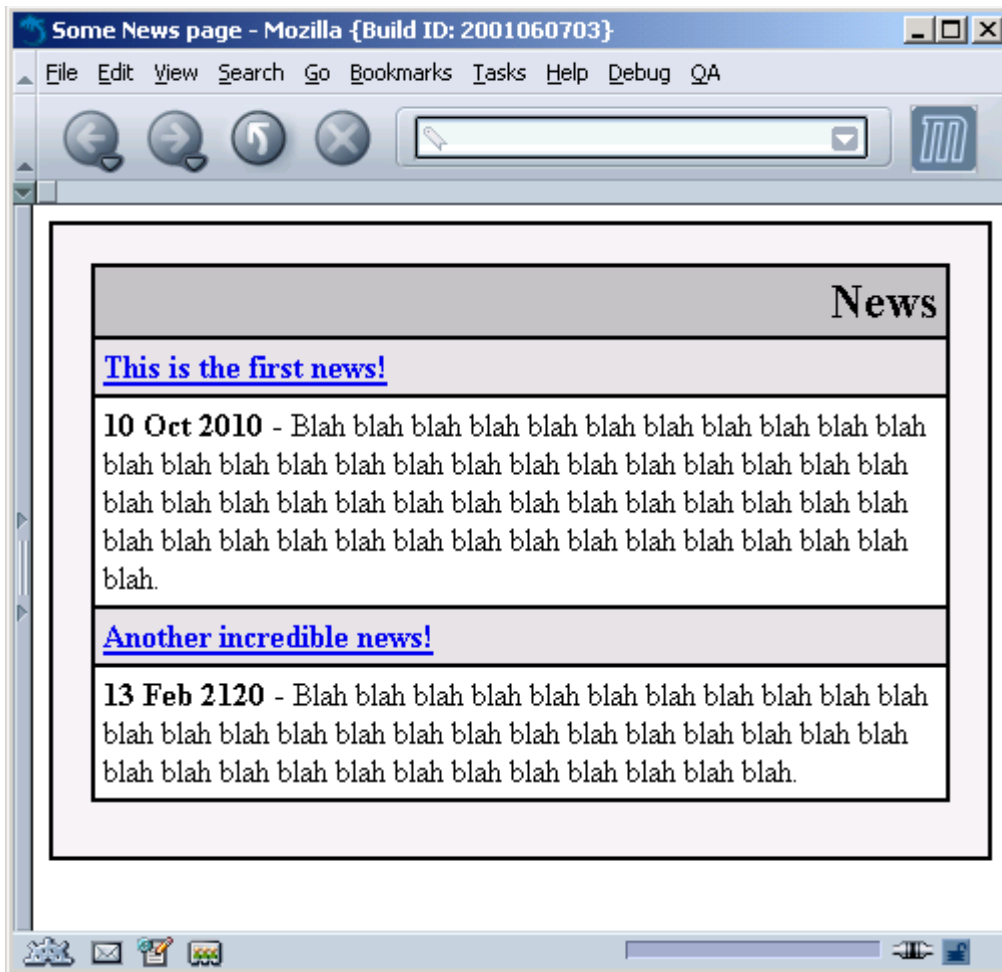


Figure 2 – Fancier News Page

The result of forcing different people to work on the same file is that more and more time is spent on managing the collisions due to concurrent work on the same information: the more workforce is added to the job, the more energy is wasted in managing collisions, the less productive becomes each single person.

This concept can be extended on the entire site information system: the more complex the site becomes, the more people it needs to manage, maintain and extend it, the more energy is wasted on managing the collisions.

If no management or architectural system is placed to coordinate the work of the different skilled groups, the entire site may reach a point where *marginal productivity* (the productivity of a new person added to the stuff) reaches zero.

This effect is what we indicate as *growth saturation*.

The graph above explains the concept in more detail: growth starts to saturate as soon as adding more people reduces the productivity of each person. In fact, the marginal productivity can be defined as the derivative of the *people/size* curve and it tends to zero as the number of people grows to meet new production requirements.

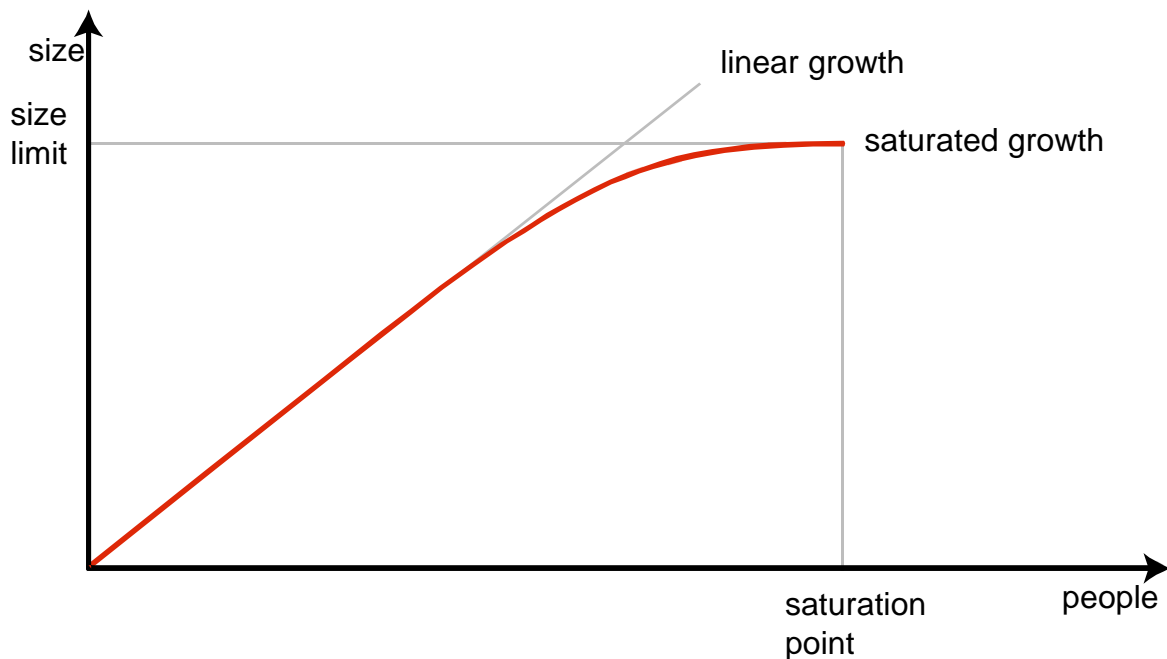


Figure 3 – Growth Saturation

### **Understanding the Reasons Behind Growth Saturation**

To understand the reasons behind growth saturation, we propose a simple model where the work activity of each individual is distinguished in two efforts: direct and indirect.

The direct production effort is the work energy that is transformed into the product and makes it possible to produce it. The indirect production effort is everything that is not directly related with the product (it's not directly included in the finally shipped product) but it's necessary for production to take place.

A saturating trend appears when the addition of new people increases everybody's indirect effort.

We now introduce a simple model to show how these saturating trends appear: first, we simplify the model by supposing that each individual that participates in production works for the same amount of time. This time is spent in both direct and indirect production efforts.

In order to be able to accomplish its job, each individual requires information from its environment and obtains it by sending messages to coworkers which returns a response only if they know the answer.

The individual that sends a message is stalled by the lack of information and cannot do anything before the necessary information is obtained.

On the other side, the receiver of the message may ignore the message after having spent a fixed amount of time reading it or spend a fixed amount of time processing it and returning the information. Only one person is responsible for processing each message.

Given a working environment of 8 hours/day total work time per individual, 1 minute the average time required to read a message, 10 minutes to process it, we analyze different people/size graphs depending on their behaviors.

### **Environment with Message Broadcasting and High Overlap**

In such an environment, each individual doesn't know who is responsible for processing his

messages and for this reason it needs to broadcast them to everyone in the group. Moreover, the technology used for production imposes a high degree of overlap where the number of messages grows linearly with the amount of people involved.

The following equation yields the amount of minutes that each individual can dedicate to direct production given the number of people in the working group:

$$Dt = 8*60 - 1*(n-1)*n - 10*n - 10*(n-1)*n/n$$

$$Tot Dt = Dt * n$$

where

- 8\*60 = minutes of total work time
- 1\*(n-1)\*n = time spent reading received messages
- 10\*n = time spent idle awaiting for message responses
- 10\*(n-1)\*n/n = time spent responding to messages

the following table and graphs show the values for *Dt* and *Tot Dt* given *n*

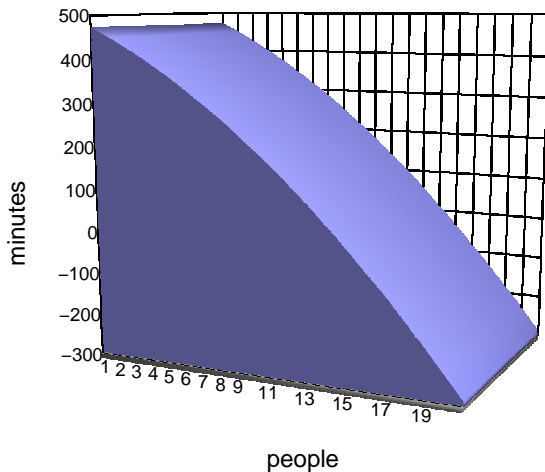


Figure 5 – Direct Time per Individual

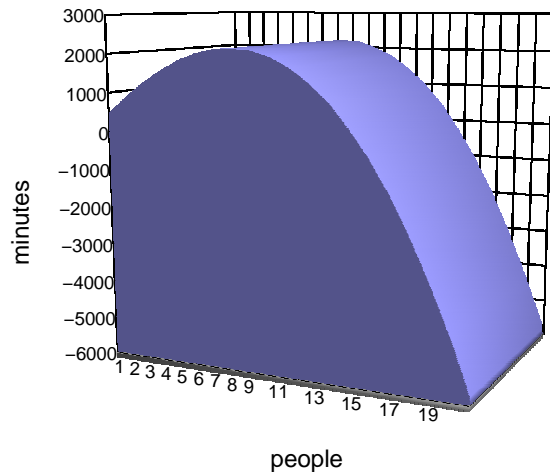


Figure 4 – Total Team Direct Time

which indicates that such an environment has a *saturation point* of 15 people and the maximum direct time of the team (thus the maximum in productivity) is reached with 8 people.

Even if this is just an example given with hypothetical environment parameters, there is evidence that such a working environment is intrinsically destined to early saturation and cannot locally scale.

### Environment with Message Broadcasting and Limited Overlap

Like in the previous environment, each individual doesn't know who is responsible for processing his messages and still needs to broadcast them to everyone in the group. However, in this case, the technology used for production enforces better separation between the work done by the individuals and the number of messages grows only with the square root of the people in the team.

The following equation yields the amount of minutes that each individual can dedicate to direct production given the number of people in the working group:

$$Dt = 8*60 - 1*(n-1)*\sqrt{n} - 10*\sqrt{n} - 10*(n-1)*\sqrt{n}/n$$

$$Tot Dt = Dt * n$$

where

- 8\*60 = minutes of total work time
- 1\*(n-1)\*sqrt(n) = time spent reading received messages
- 10\*sqrt(n) = time spent idle awaiting for message responses

$$10*(n-1)*\sqrt{n}/n = \text{time spent responding to messages}$$

the following graphs show the trends for *Dt* and *Tot Dt* given *n*

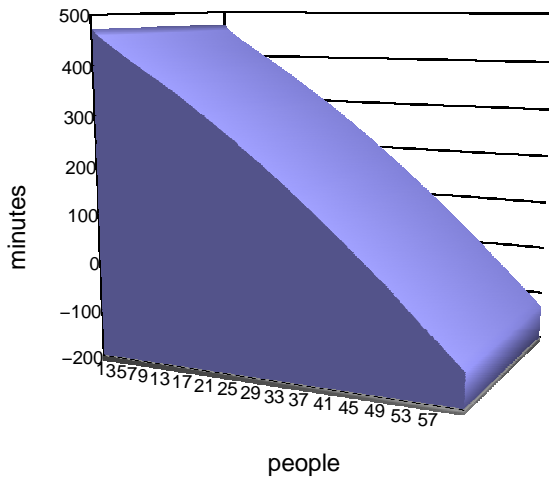


Figure 6 – Direct Time per Individual

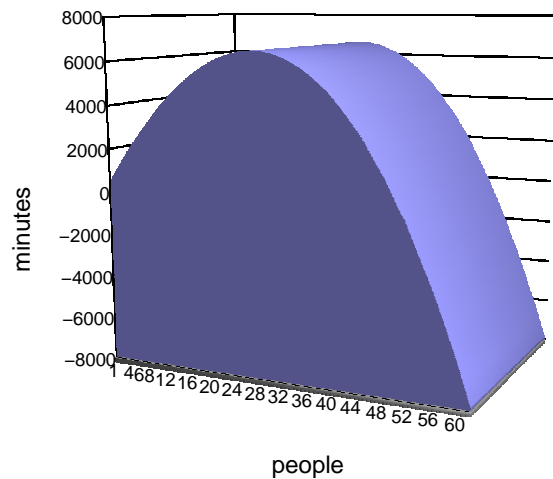


Figure 7 – Total Team Direct Time

which yields a *saturation point* of 49 people and a *maximum direct time point* of 27 with the exact same parameters (thus individual productivity) of the previous environment.

### Environment with Direct Messaging and No Overlap

Unlike the previous environments, each individual knows who is responsible for processing his messages and doesn't need to broadcast them to everybody. Moreover, the technology used for production enforces a complete separation between the work done by the individuals and the number of messages doesn't depend on the team size.

We assume an average of 5 messages a day as an example.

The following equation yields the amount of minutes that each individual can dedicate to direct production given the number of people in the working group:

$$Dt = 8*60 - 1*(n-1)*5 - 10*5 - 10*(n-1)*5/n$$

$$Tot Dt = Dt * n$$

where

- 8\*60 = minutes of total work time
- 1\*(n-1)\*5 = time spent reading received messages
- 10\*5 = time spent idle awaiting for message responses
- 10\*(n-1)\*5/n = time spent responding pertaining messages

the following graphs show the trends for *Dt* and *Tot Dt* given *n*

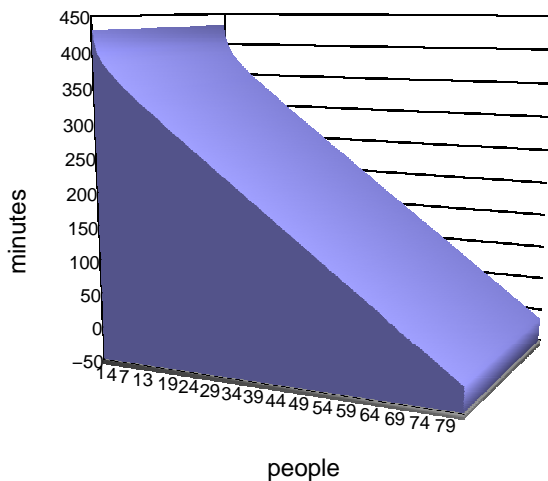


Figure 8 – Direct Time per Individual

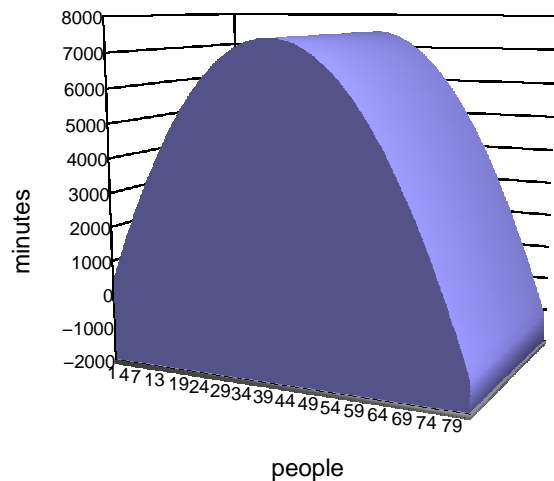


Figure 9 – Total Team Direct Time

which yields a *saturation point* of 78 people and a *maximum direct time point* of 39.

### The Economical Aspect

The figures presented so far don't present a direct economical value, for this reason we now add a simple economical aspect to the model by supposing that each individual costs 100\$/day. The results are summarized in the following table:

	Env 1	Env 2	Env 3
<b>Maximum Direct Time Point</b>	8	26	38
<b>Maximum Direct Time</b>	2192	6565	7460
<b>\$/min at MDTP</b>	2,92\$	10,3\$	19,36\$
<b>Direct/Indirect time % at MDTP</b>	57.08%	52.60%	40.90%

Table 1 – Summary at MDTP

Assuming that production power is function of *direct time* only, the first we note is that the third and optimal environment is roughly 3,5 times more productive than the first, but it costs 5,6 times more, while the second is almost 3 times more productive but it costs 3,5 times more.

Another interesting aspect is the percentage of time spent on direct production which, rather counter-intuitively, *lowers* with the messaging efficiency.

But presenting data in such table may be misleading: in fact, we are comparing the values at MDTP. The following table compares the three environments at the same people number which is the MDTP for the first one:

	Env 1	Env 2	Env 3
<b>People</b>	8	8	8
<b>Total Direct Time</b>	2192	3257	2810
<b>\$/min</b>	2,92\$	1,96\$	2,28\$
<b>Direct/Indirect time %</b>	57.08%	84.83%	73.18%

Table 2 – Summary at 8 people

The picture changes rather significantly: reducing overlap and decreasing the necessity for people to exchange messages in order to accomplish their work has boosted their productivity by 48%.

### ***Conclusions on Growth Saturation***

Even if the model presented is very simple and can rarely apply *as is* to reality, we believe it's useful to show how growth saturating trends appear no matter how efficient and organized a working group is.

We believe this is intrinsically tied to the nature of the *task-force* organicistic model [Law73] and shows an evident parallel with the saturating trends shown by peer-2-peer systems [Rit00].

But the most important aspect shown by this analysis is the importance of reducing the amount of synchronization required when there is overlap and information must be exchanged between individuals to coordinate their access to the concurrently used resource.

# An Optimization Strategy

## Divide et Impera

The model chosen in the chapter above to describe a team of coworkers highly resembles those found in the description of computer networks. It is, in fact, common knowledge that a *completely connected* network is economically unfeasible because the number of connections grows with the square of the number of nodes, or precisely, as  $n(n-1)/2$ .

A good parallel can be drawn between such a model and the *ethernet* network technology. In such a widely popular technology used to connect *local area networks* of computers, each node transmits data packets of fixed length over the wire when it is required to do so. If the wire is not used (means: no packet is being transmitted) the packet is received and all is well. Otherwise, a *collision* is detected and the two have to retransmit the packet after a random delay to reduce the case of further collisions.

Such a technology can be modeled with the same *completely connected graph* that we used to model our working environments. In fact, as all system administrators know very well, the number of computers that can be directly connected to an ethernet cannot exceed a few decades (30–40) without suffering extreme performance degradation (or even total loss of transmission).

In our model, collisions were represented by the necessity of working on a shared resource, which required exchanging information and creating delays for their processing.

The most used solution to reduce these collisions and avoid severe performance degradation is the use of special network devices called *switches* which are responsible for physically isolating one or more networks and behave as filters.

The switch is capable of knowing if a packet cannot be handled by the network where it was generated and must be *switched* and passed over to the other network. At this point, is important to note that switching does not automatically reduce the chance of network collisions, but does so only if the number of data packets to be switched is generally lower than the number of packets that are processed in the same network where they were created.

In fact, switches are effective if they *partition* a network in more subnetworks that have a higher chance of requiring communication (normally departments or rooms or floors).

This strategy reduces collisions since the switch physically isolate the two networks and if two transmissions happen in the two networks and are handled by the same network where they were generated from, the switch remains silent and the two networks can accomplish transmission in a totally parallel way.

On the other hand, the two networks are physically separated, but logically connected, thus allowing the establishing of communication between the two networks transparently, as they were a single network.

### Partitioning by Concern

The strategy used in the ethernet case can be abstracted using the software engineering concept of *separation of concerns* (see Appendix A for more details).

The act of partitioning a network in subnetworks where higher chance of requiring communication is encountered can be more generally interpreted as the act of partitioning a given space into *concern islands*.

A *concern island* is a set of entities that share the same concerns.

For this reason, the entities that belong to a given concern area are much more likely to require interaction with other entities in the same concern island rather than entities that belong to other islands, thus exhibiting different concerns.

SoC is a concept that is normally applied in software engineering to allow systems to be componentized, minimize cross-talk and standardize the ways components collaborate to create the system.

Noting the parallel between the topological solution used to reduce the connections in fully connected graphs and the concept of modularization, we suggest the use of the same optimization strategies applied to human resources, physically isolating people that share the same concerns but allowing logical connection between these concern islands thru the use of *peers* that are responsible for routing out of the concern island those connection requests that cannot be handled inside the concern island.

It must be noted, however, that by *physical isolation* between concern islands we *do not* mean that people are physically located in different rooms, floors or buildings, but that the work done in one concern area can be performed without requiring intensive interaction between other concern islands.

While it is common practice to physically isolate working groups with different skills and concerns, the importance of analyzing if their work is truly isolated is generally overlooked and its impact on productivity and general work quality greatly underestimated.

For this reason, we believe that the concern analysis and the engineering of their separation must be performed both at the human level (skills, capacities, attitudes) but also at the underlying technological infrastructure, especially in high-tech working environments such as those required by the creation of complex web sites.

## Concern Overlap and Mismanagement Bottlenecks

Just like the use of switches doesn't increase network performance by itself, but it requires a wise network partitioning that reduces the need for intra-subnetwork communication, concern overlap is responsible for forcing one person in one group to require to communicate with another in another group.

The equivalent of switches in our working group model are *project managers* (or project coordinators) which act as connectors between working groups and are responsible for that group with the rest of the company.

Project managers should be responsible for *logically connecting* the different groups together and to *route* those incoming messages to the correct people that should (or simply can) process them.

While it is pretty easy to partition a company in divisions or groups which are well separated and well balanced in the number of people and managers involved, it is generally ignored (or underestimated) the impact of technological constraints used for production and how this technology might impose concern overlap that are not reflected in the organization graph.

When such thing happens, the number of messages that need to be exchanged between groups is very high and since it's forced to pass thru the manager, it may soon become a bottleneck for the

work of the two groups and might be bypassed in order to required the necessary bandwidth between the two groups.

Since these circumstances where managers become bottlenecks are easily observable, it is easy to blame the manager's inability and to believe that the problem is personal and not structural.

On the other hand, we have shown how a perfectly balanced organization diagram might be misleading since the real separation of concerns happens at technological level and it's quite common that this remains unnoticed at higher management levels.

In fact, having a technological architecture that forces concern overlap between different working groups not only exhibits bottlenecks at group management level, but also shows an interaction diagram that tends to the topology of *fully connected graphs*. In this case, the entire company exhibits a saturation trend that follows those described in the previous chapter and thus show that even if perfectly managed (which is normally not the case if high overlap exists) the productivity cannot scale above a certain point.

## Framing Concerns

We have reached an important result: one of the possible strategies to avoid the effects of growth saturation is the application of separation of concerns at the organization level.

But we believe this is generally useless if not backed up by an equivalently well-balanced separation at technological level which is responsible for allowing different working groups to operate in parallel without requiring synchronization on concurrent resources or frequent updates on their status.

We also believe that the lack of such separation enforcement at technological level is responsible for establishing interaction requirements which are very similar to a fully connected graph, thus making management perceived as unnecessary (since communication between groups tend to happen at horizontal level to increase communication efficiency) and productivity exhibits the saturation trends very similar to those exhibited by single and fully connected groups.

It is interesting to note that growth saturation starts to be perceived as a company requires more productivity and starts employing new people. When this happens, the company organization diagram is already established, as well as the technological assets. This is because the need for an increased productivity reflects a previously successful behavior of the company, so that new products/projects are to be developed.

It is at this point (normally when the *maximum direct time point* is reached) that the saturation effects become effective and start reducing significantly the amount of production power and requiring more people (and more expenses) than previously accounted for.

When this happens, it is generally considered more economically feasible to continue adding more resources to production rather than reconsidering how the work is organized both at management and technological level.

In those cases where high management is capable of perceiving the effect of growth saturation and propose architectural changes, they are normally focused at the organization level (repartitioning the working groups, changing managers, reassigning duties) rather than at the technological level.

Moreover, in those cases where the technological level is conceived as an architectural limit to productivity growth, technological changes are normally not reflected by proper organization changes and may turn out to be totally ineffective due to the lack of synchronization between the two aspects.

For this reason, we reach the conclusion that in order to avoid the problem of growth saturation, both the technological and the organizational architecture must be adapted to productivity needs and to the number of people involved.

This said, it must be noted how design ideas behind current web technologies almost never take these conclusions under consideration: this results in a continuous mismatch between organizational solutions and technological solutions and leads to a permanent feeling of mismanagement in production groups that become successful and need to scale.

It is obvious, at this point, to conclude that it is necessary to design a technological framework that is centered around these abstract considerations and provides a way for the two levels (technological and organizational) to work together and improve overall production scalability.

# Framework Design

## Applying SoC to Web Publishing

We have seen how the application of software engineering concepts like *separation of concerns* to the engineering of workgroups organizations can be used as an optimization strategy to reduce the effect of wasted production energy and limit growth saturation.

Even if we believe this is a general concept and can be applied to almost all production environments, we now focus our attention on web site production and outline the design of a technological framework that can implement that separation at technical level which we consider necessary in order to reduce the effect of growth saturation.

The principles of SoC applied to human resources indicate that each individual exhibits different concerns that are normally associated with its skills, attitude or past experience.

In order to allow the separation of these concerns into different workgroups, the *problem space* must be analyzed and the concerned required for the solution of the problem indicated and explained.

Then, each *concern island* must be connected with the others with appropriate *contracts* which must be backed up by appropriate technological solutions.

Our analysis will focus on the production of medium/big web sites where information clustering is evident and therefore far from the original web development model of individual information owner and maintainer.

### The Flat Concern Layout

A first and rather simple analysis divides web site production in three major concern islands depicted in a flat hierarchy and driven by skill separation:

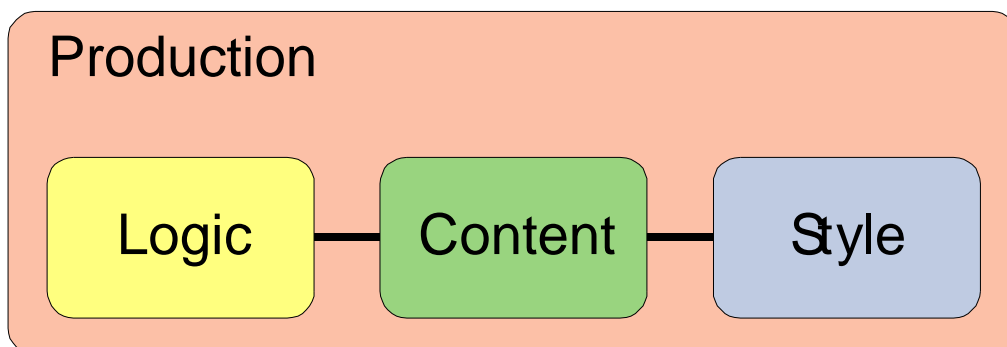


Figure 10 – The Flat Concern Layout

### Content

The content island gathers those concerns that regard the authoring and editing of all the human created information that is published.

This information can be text, audio, video or all sorts of media that requires direct human

intervention for its creation. An example of such content is the text contained into an article, while a news-feed that is retrieved from an external resource and does not require human intervention from this side must not be considered content but rather raw data that is available to the system.

The typical skills required in such concern island involve the capacity to author, edit, translate, adapt or otherwise process or create information to be published.

In the original web model, the people working in this concern island are the equivalent of the content owner, which was responsible for both creating, maintaining and publishing its information.

In bigger sites, they are responsible for the creation and maintenance of the internally human created content but should not be responsible for their publishing since it involves skills and concerns that belong to a different concern island.

The same can be said for all the raw information that is available to the web site but does not require direct human editing.

## **Style**

The style island is responsible for presenting the information that has to be published.

This includes the creation of all the art work and visual appeal that give both recognition to the site and a pleasant look to the published information.

The artistic concerns of publishing involve completely different skills than those required to author the published information and for this reason they should be kept completely separate.

In fact, those skills mainly involve drawing, esthetic sense, color matching, knowledge of human perception and everything that requires a direct interface to the human user.

## **Logic**

The logic islands is responsible for all the automatically generated information that can be either directly published or given as raw information for the content authors.

It is also responsible for implementing all the technological details that allow user interaction, data generation, data storage and replication and all the technological details that support the act of publishing in general.

The required skills are purely technological, ranging from direct programming to system integration.

## ***Crosscutting Concerns***

Even if this partition of the problem space is simple, attractive and would represent itself a major improvement over the old web model, it has a major drawback in its flatness: each concern area is responsible for an important part of the production, but none is on top of the other.

The tree production areas operate to produce what's required for the web site, each one in its specific context, but there is no concern area that has a global view and drives the publishing process.

A symptom that the above concern layout is not sufficient can be identified in the number of *crosscutting concerns*: these represent concerns that are shared between different islands and all of them can claim jurisdiction on the problem.

The presence of crosscutting concerns can rarely be avoided, but a good concern layout must try to

eliminate the overlap between concern islands especially if it involves important production problems.

And this is the case with the above model, in fact, a number of problems cannot be automatically placed into one of the three areas:

### **Design and Maintenance of the URI Space**

The importance of a good design of the URI space is generally underestimated but modern and scalable web sites cannot avoid spending resources on this topic (see Appendix C for a deeper analysis). Unfortunately, this is a realm that cannot be easily applied to the flat model proposed above.

In fact, both the content and logic areas can claim jurisdiction on the problem. The first because hyperlinking is generally considered part of content authoring and the second because the URI space is generally managed by the web server, thus the technological infrastructure.

The implicit mistake here is that the URI space represents one of the strong contracts not only between the different concern islands responsible for the site production, but also with the users and with all the other sites on the web.

### **Content Aggregation**

With the flat layout we have identified two different types of information: *raw* (which means automatically generated or acquired from external resources) and *cooked* (which stands for information that is created by direct human intervention).

While information is more often divided in *static* and *dynamic*, we believe that with the advent of complex web technologies, the difference between static and dynamic doesn't necessarily mean human created and machine generated as it used to be.

However, even this distinction between human edited and machine generated information is not enough since as site complexity grew, published information started to be a mix of the two types that were aggregated by the publishing system following some layout indications.

Following the newspaper model where the journalist is not responsible for creating the page layout but only the content of the articles, it is intuitive that editors and page composers have different concerns and should be hierarchically distinguished.

### **Flow Design**

The web started as a GET-only media where people requested resources and servers served them, but the capabilities of HTTP to POST information to a resource allowed the creation of more complex interaction.

Such interaction is identified by the *resource flow*, a diagram that indicates the logic connections between resources along with the different states that they can have.

In the flat model, the skills required for flow design and the concerns associated to it are equally spread between the three groups. In fact, the content island might want to control the flow because they consider it directly associated with the content published, the logic island finds it natural to control and design the flow because intrinsically driven by the publishing system and the style island might want to influence it as well because of esthetic reasons such as pagination or readability.

## ***An Improved Concern Layout***

The list of the crosscutting concerns found in the flat concern layout outlines these problems:

- A flat layout is unable to drive efficiently the production process because it doesn't remove overlap for a number of important activities.
- Simply adding a management group on top to guide these overlapping activities doesn't solve the issues because it doesn't separate the concerns and it's very likely to reduce efficiency by creating bottlenecks rather than improve it.
- There is no central authority that creates and modifies the contracts and has sufficient technological ability to shape the technological layer to match the management requirements.

For all these reasons, it becomes natural to extend the original model by adding a new concern island that we identify as the *design island* that is responsible for establishing the skeleton and the structure of the site and for establishing and controlling all the contracts that allow the other three islands to work independently.

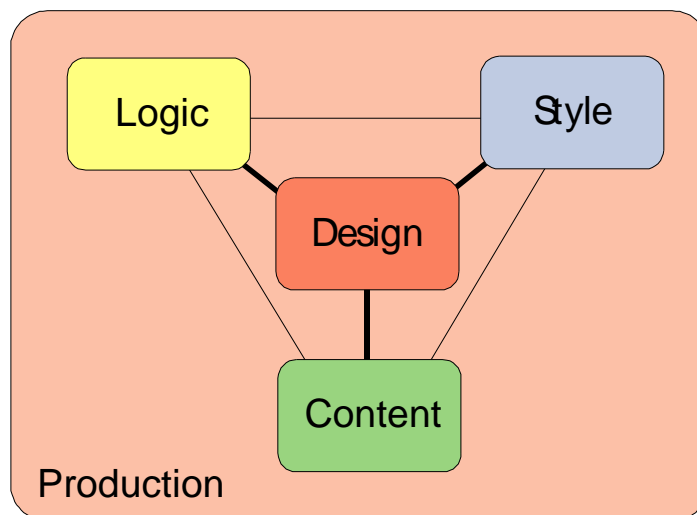


Figure 11 – Hierarchical Concern Layout

With this new layout, those issues that could not be solved without dealing with concern overlap, can now be fully dealt with without requiring any concern island to work concurrently or fight over control of the contracts.

In fact, this *design island* is not only responsible for creating the contracts and maintain them, but should also guide the integration process between the work done by the other islands below.

With the introduction of such concern layout, not only a natural hierarchy is established, but it allows the overall production activity to be able to concentrate on vital issues such as *usability* which naturally crosscut all concern islands.

Therefore, the skills required in the *design island* should be a mix of all the skills required to understand, design and implement the global publishing requirements. Thus it must include both usability experts, system architects and content redactors.

## ***A single marketing–production channel***

So far we have outlined the separation of concerns required for the actual production of the site and we haven't dealt with the purely economical aspects of the problems.

With the introduction of a *design island* on top of the other three, we have created a natural

communication channel between the production area and the marketing area.

With a flat hierarchy, there is no single communication channel between marketing and production. This fact not only makes it harder to communicate ideas, issues and solutions between the two areas, but also increases the amount of overlap between the concern islands of the production area because there is no single referent.

This is a very important concept: both the economical and operational aspects are necessary for the maintenance of the site but represent different souls and normally go in different directions.

A vivid example of this is the need for more advertising space required from the economical side and the need of more pleasant user experience from the design side. In this particular example, it is interesting to note that a more pleasant user experience is very likely to increase visits, thus allowing the site to increase its overall advertising space as well as its visibility, but these concepts are rarely adopted because of the difficult communication between the two areas and the lack of a single communication channel between the two.

### ***Limitations of This Approach***

We have seen the benefits of an intrinsic hierarchical approach, but it also exhibits some limitations:

#### **Difficult reshaping of existing structures**

It is easy to imagine that existing flat structures will have a hard time accepting a new structure where they automatically lose their power and visibility at the expense of another level of indirection. This is evident especially considering that all the design decisions (which are normally considered the valuable part of the production) are removed from the existing islands.

#### **Necessity of wider skill diversity**

While the flat structure is inherently focused on separating people with different skills, rather than reducing concern overlap, the introduction of the new design island fully separates concerns but forces the group to be composed by people with skills previously found in different groups.

### ***The Necessity of a Technological Foundation***

So far we have identified a concern layout that allows all of the important web site production problems to be easily associated to a precise concern island.

But it must not be forgotten that architecting separation of concerns at the organization level is not enough to overcome the effects of growth saturation unless a solid technological foundation permits such concern separation to remain so in practice.

This is the key point: a work organization that doesn't match the way the people actually operate and interact is not only an academic exercise, but also increases costs by forcing people in a structure that doesn't follow the natural way they would work if free to organize themselves.

This is somewhat equivalent to the chicken/egg problem: the work gets organized to allow people to work the way they find natural for their skills and background. Unfortunately, the technology around web sites wasn't designed for teams but for individuals and it comes out that some major issues (that we identified as design's concerns) don't appear naturally important because web technologies don't guide the work in that direction.

Seen from this angle, it appears evident that in order for the proposed work organization to function and appear natural to the people involved, the underlying technologies must be designed in order to

give the proper result to these issues. Only in this case, a work organization based on separation of concerns and well defined (and technology driven) contracts can both appear natural to the people that make up the workgroup and reduce the effects of growth saturation.

# Enabling Technologies

## The First Step Toward Implementation

The concepts outlined so far would remain purely academic if not backed up by a software implementation.

A few constraints were identified:

- **Portability:** in order to achieve the maximum possible range of uses, as well as to maximize reuse of existing technological facilities, the software should not be tied to a particular hardware or operating system or commercial software.
- **Interoperability:** the implementation should allow interoperability with both existing technologies (back compatibility with legacy applications) as well as providing future compatibility.
- **Extensibility:** the software should be extensible in order to provide a mean of personalization or enhancement without compromising the other constraints.
- **Performance:** the implementation should be powerful enough to replace existing web publishing solutions even for very high end requirements. For this reason, it should be fully scalable and carefully avoid scalability constraints.
- **Simplicity:** the implementation should be as simple as possible, yet without compromising the rest of the requirements.

### **Software Platform**

Since portability is one of the key requirements, it was decided to write the implementation on the Java Platform. Java [Java] is a strongly typed object oriented language with provides many modern features (such as automatic memory management, exception handling, etc...) and is based on a very powerful set of *application programming interfaces* (API).

But the major portability benefits of the Java Platform are given by the fact that Java source code is compiled into an intermediate binary format (named Java *bytecode*) which is then interpreted by a Java *virtual machine*, an executing environment responsible for adapting the binary code to the native platform underneath.

Such architecture allows complete separation of concerns between the software writers and the virtual machine implementers: in fact, by coding against the standardized Java bytecode and against the standardized set of APIs that the Java Platform provides, a compiled program can be executed by every compliant Java Virtual Machine.

These virtual machines exists for the vast majority of currently used operating systems, making it possible to completely eliminate the need for complex compatibility workarounds that plague other languages such as C or C++.

### **Web Serving Environment**

Another big portability issue comes from compatibility with web serving environments (such as

web servers or application servers): the implementation of a web publishing framework should not require the replacement of currently used technologies but rather integrate seamlessly with existing solutions.

For this reason, such a publishing system should not replace but rather *extend* existing functionality. Therefore, it should be implemented as a module to existing web serving environments.

The choice went for the use of the Servlet API [Servlet/2.3] as the modular interfaces to connect to existing solutions. These APIs were originally designed for the Java Web Server, a pure-Java web serving solutions developed by Sun Microsystems, but were ported to all other web containers with the creation of *servlet engines*, server-specific modules that make it possible to run *servlets* transparently in any web server.

The Servlet API model creates a common denominator for Java server applications to connect to web servers in order to receive client requests and be able to generate responses.

Servlet engines exist for almost the totality of web serving solutions and thus provide, together with the portability of the Java Platform, the possibility of transparent portability across the entire range of existing web serving environments.

Servlets are, in fact, a way to extend web server's functionality in a completely portable way.

## **Data Description Languages**

A publishing framework is responsible for doing all the processing required to publish some information on the web with the requested data format.

For this reason, the data required by the framework (along with the content to be published) must be described by a language that is expressive enough to allow contextualization of the information described.

## **The Limits of HTML**

The biggest limit of the HTML language is its presentation-oriented nature: in fact, HTML uses markup mostly to add graphic and visual information to the included resources (text, images, video, etc...). Web browsers are computer programs that read that visual information and use it to display it on a screen.

An example for all: the `<b>` tag indicates that the text included in between these tags should be rendered with a bold font face. While this doesn't create any problem when visualizing the information on a computer screen, it doesn't have the same meaning when the information is rendered thru a voice synthesizer. What does *bold* mean for voice? It means *strong* and in fact, in later HTML versions the tag `<strong>` was introduced to create a more semantically meaningful markup.

At first this might appear as a linguistic detail, but it's just the tip of an iceberg of semantical incorrectness which surfaces more evidently when we want to search large amount of HTML-encoded information.

Being HTML visually oriented, the real semantic meaning of the information described cannot be algorithmically extracted but must be heuristically guessed depending on visual properties (such as a bigger font, a centered text, or location on the page).

There is no way to contextualize searches because there is no way for a machine to *understand* what the page *means*, even in a very simple and mechanical form because the only contextual information included in HTML pages along with the text is how to draw the page on a screen.

HTML includes at least other two semantic information: hyperlinking and metadata. While the first is vital for the creation of the web (and mainly the reason for the establishment of HTML), the second was introduced to improve the data mining experience in large sets of HTML data thru the use of keywords which are used by metadata-aware crawlers to index the pages in a more significant way.

While keywords and hyperlinking topology analysis provide a step forward in the searching experience (as the Google search engine shows), the HTML model is intrinsically limited by the fact that is not extensible by design.

In fact, the HTML specifies that interpreting programs must ignore all markup that they don't recognize, without even triggering a warning or an error to the user.

This allowed browser vendors to create specific markup that is meaningful on their platform only, starting what became known as the *browser war*, the attempt of software corporations to create proprietary extensions and lock users into their platforms for monopolistic reasons.

But extensibility is a feature that was requested by many, starting from the scientific community itself, which wanted to be able to add mathematical markup to HTML to allow scientific papers to be easier to publish (math formulas are currently rendered as images and included in the articles, but this is a very poor use of the technology).

Other proposals for extensions included markup for vector graphics, for chemical formulas, for genetic sequences. The web architects soon understood that making HTML extensible was simply a step in the wrong direction: it would have started the creation of different flavors of HTML that worked with different users agents, making both the user experience and web publishing more expensive in terms of time, frustration and authoring costs.

## A Step Back Two Steps Forward

Since HTML borrowed the SGML syntax, it was an obvious choice for web architects to take one step back and instead of making one language extensible, allowing the creation of a common set of technologies, independent on the semantics of the language, making it possible to extend the semantics of the digital resources published on the web, but without fueling incompatibilities.

Soon they realized that SGML and related languages were simply too complex and allowed many features that the web did not require. The World Wide Web Consortium (W3C, the organism responsible for the maintenance of the web specifications) started an SGML-4-the-web working group that aimed to create an SGML subset that could become the foundation of all the markup languages on the web.

The result of this effort was called *eXtensible Markup Language* (XML) and was recommended in 1998 and started the creation of a series of related technologies that aim to create a new foundation for web languages to extend the HTML model without incurring in its early design faults.

## The eXtensible Markup Language

The *eXtensible Markup Language* [XML] is today generally considered the best choice for a data description language, along with the use of *XML Namespaces* [XMLNS][Cla99], for its unique features which can be mainly listed as:

- **Internationalization ready:** XML is based on Unicode, a 16 bits language-neutral character

encoding which is future compatible with all modern languages. This not only guarantees future extendibility but allows transparent geographic portability.

- **Character based:** XML is based on characters rather than binary strings and it's therefore editable with the use of any text editors and doesn't require special treatments. This removes the need for specific authoring tools and allows the data files to be directly readable.
- **Standardized:** XML is a recommendation of the World Wide Web Consortium (W3C) and is supported by the vast majority of entities in the information technology field, both commercial and academic.
- **Multidimensional:** the addition of namespaces to the XML syntax allows markup to be fully multidimensional, in the sense that different markup vocabularies can be mixed yet maintaining their uniqueness by their association with their namespace identifier.
- **Fully extensible yet validating:** unlike specific markup languages such as HTML, XML is not a language but a more properly a syntax that indicates how data can be encoded in such a way that content and markup are fully separated. It does not specify the markup that can be used, but it does specify a way to *validate* if a specific XML file follows specific rules.

There are two level of compliance with the XML specification: a file might be *well-formed* if it follows the XML syntax but no *document type definition* (DTD) of the markup is available; or the file might be *valid* if it is well-formed and it follows the rules indicated by its associated DTD.

The following is an example of a well-formed XML file:

```
<?xml version="1.0"?>

<page>
  <title>Hello</title>
  <content>
    <para>This is my first XML file!</para>
    <empty-tag/>
  </content>
</page>
```

While the following shows an example of a *valid* XML document:

```
<?xml version="1.0"?>

<!DOCTYPE page [
  <!ELEMENT page (title?, content)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT content (para+)>
  <!ELEMENT para (#PCDATA)>
]>

<page>
  <title>Hello</title>
  <content>
    <para>This is a valid XML page!</para>
  </content>
</page>
```

where the highlighted text represents the *document type definition* which is responsible for indicating what elements and attributes are considered part of this language and how they can be used and nested.

While the above examples do not make use of namespacing facilities, here there is an example of this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<page:page
  xmlns:page="http://mysite.org/page"
```

```

    xmlns:graphics="http://mysite.org/graphics">
<page:title>A Simple Namespaced Page</title>
<page:content>
  <page:para>This is a textual paragraph</page:para>
  <page:para>Below you see a graphical rectangle</page:para>
  <page:figure>
    <graphics:rectangle x="100" y="200" width="20" length="40"/>
  </page:figure>
</page:page>

```

In this example, two namespaces are used (<http://mysite.org/page> and <http://mysite.org/graphics>) which are used to identify two different markups, one for page description and another for vector graphics description.

Multidimensionality becomes important as the complexity of the framework requirements grow. In fact, the namespacing facility of XML allows the creation of specific markups which can be used together when it makes sense to do so.

In this simple case, the *figure* element contains graphic information to draw the figure, while in other cases, might be required to link a figure from an external image source. Namespace do not only make it possible to associate a markup with a specific URI, but also make element names unique by following the intrinsic uniqueness of URIs. This avoids names collision for elements and attributes.

This fact has a great impact on both modularity and portability because it allows different markup languages to be *composed* without requiring proprietary changes that could limit portability and increase the risks of having to maintain and enhance these proprietary changes.

## Transformation Languages

While HTML has many limitations as a data description language, it has the intrinsic capabilities to indicate how the information must be presented to the user. The HTML source code, in fact, remains hidden behind the browser's curtains during the entire web browsing experience, while the browser presents the information contained in the page to the user in the way that the resource authors wanted.

This model fits perfectly the case were a single individual is both the owner of the content and the one responsible for its presentation. If this is normally the case for homepages or small collections of documents, this is almost never the case for bigger sites where content editors and graphic designers are normally different people.

The web architects soon realized that this *concern overlap* was going to be a serious scalability limitation for the web model and proposed the creation of *stylesheet languages* that made possible to separate content and graphics on different files and give their ownership and responsibility to different individuals.

The W3C has standardized two of such languages, *Cascading Style Sheets* [CSS2] and *eXtensible Stylesheet Language for Transformations* [XSLT].

Both allow complete separation between data and its presentation information, but XSLT is a much more general language, allowing, in fact, to transform an XML document into another with completely different semantics.

XSLT is itself a markup language that uses the XML syntax. Here is an example:

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

```

```

<xsl:template match="page">
  <html>
    <head>
      <title>
        <xsl:value-of select="title"/>
      </title>
    </head>
    <body bgcolor="#ffffff">
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="title">
  <h1 align="center">
    <xsl:apply-templates/>
  </h1>
</xsl:template>

<xsl:template match="para">
  <p align="center">
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="empty-tag">
  <!-- do nothing -->
</xsl:template>

</xsl:stylesheet>

```

From the example, we see how every XSLT stylesheet must be a well-formed XML file and they use namespaces to separate the transformation markup (which in this examples uses the *xsl:* prefix) from the content markup that is placed into templates.

XSLT is a declarative language and it's based around the concept of *templates* which are matched against XML Paths [XPath]: the XSLT processor scans the XML document to transform starting from the root element and every time a matching XPath is found, the associated document is applied in that location.

Transforming our first XML example with this stylesheet yields:

```

<html>
  <head>
    <title>Hello</title>
  </head>
  <body bgcolor="#ffffff">
    <h1 align="center">Hello</h1>
    <p align="center">This is my first XML file!</p>
  </body>
</html>

```

which is a well-formed XML file using the XHTML markup (the HTML markup adapted for the XML syntax).

It is interesting to note that the content of the *title* element in the XML file has been used twice: unlike CSS which adds presentation information to the existing document structure, XSLT is capable of rearranging the location of the original element tree or even create new content procedurately.

This is the key feature of this language because it allows to use it for general transformations and not only styling.

For example, this other XSLT stylesheet

```
<?xml version="1.0"?>
```

```

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="page">
    <wml>
      <card id="index" title="{title}">
        <xsl:apply-templates select="content"/>
      </card>
    </wml>
  </xsl:template>

  <xsl:template match="para">
    <p align="center">
      <xsl:apply-templates/>
    </p>
  </xsl:template>

</xsl:stylesheet>

```

transforms the same original XML example into

```

<wml>
  <card id="index" title="Hello">
    <p align="center">This is my first XML file!</p>
  </card>
</wml>

```

which uses the *Wireless Markup Language* [WML], an XML language used by the *Wireless Application Protocol* (WAP) framework on modern cellular phones or personal digital assistants (PDAs) to present content.

## **Presentation Languages**

We have already seen how the same content can be presented using different markup languages depending on the programs that has to visualize it.

The most used presentation language in the XML world is, rather evidently, XHTML, the XML adaptation of HTML markup and semantics. XHTML is the perfect choice for all those presentation requirements that mimic today's web and it's very likely to remain the most used choice for both its simplicity and power.

At this point, we must note how, with the ability to transform any XML markup into any other, XHTML can now be used with its original design goals: as a presentation language and not as a data description language. In fact, it is up to the transformation stage to adapt the content described by a more semantically-specific markup language, transforming it into something that can be visualized directly by clients that do not recognized the markup used to describe the data.

There are many presentation languages that use the XML syntax and the most important are:

- eXtensible HyperText Markup Language [XHTML]: the XML-ized version of the HTML markup language.
- Wireless Markup Language [WML]: a presentation language for small and portable devices characterized by small screens and limited input capabilities such as cell phones and wireless portable digital assistants.
- Scalable Vector Graphics [SVG]: a vector-based graphical markup language with advanced capabilities such as procedural filters, scripting-based interactivity and complete 2D graphic capabilities.
- eXtensible Stylesheet Language Formatting Objects [XSLFO]: a markup language for advanced typesetting and paginated presentation for both digital media and paper.

- eXtensible 3D Markup Language [X3D]: the evolution of the *Virtual Reality Markup Language* (VRML) written using the XML syntax and used to describe complex three-dimensional environments, along with the user interactivity, animation, behavior, sounds, etc.
- Mathematical Markup Language [MathML2]: a markup language for describing and presenting all sort of mathematical formulas using the XML syntax.
- Voice Markup Language [VoiceXML]: a markup language for presenting content and describe interaction in voice-based environments such as automated telephone systems, answering machines, accessibility tools for the visual impaired, etc.

Each of these languages is standardized by an organization or industry consortium and freely available to the general public for use and implementation. There exist several implementations of browsers, readers, or otherwise compatible programs that are capable of understanding these formats.

### **The Babel Problem**

With the availability of software that understands these presentation formats and the ability to transform any XML markup into these widely recognized formats, we are now protected from the influence of the *babel problem*.

This problem was raised as a critique to the generality of the XML language when it was first introduced. In fact, even if severely limited in its semantics and expressive capabilities, the HTML language was *de-facto* recognized by all sorts of devices and applications. With a more generic language, each individual or organization could, in theory, come up with their own proprietary semantics, creating the so-called *babel effect* where the web could be *balkanized* into several regions depending on the markup dialect used.

With the creation of a powerful transformation language well integrated into the XML technology framework, the risk of fragmenting the web is somewhat reduced, allowing data to be marked-up with whatever language is found appropriate, yet allowing it to be *adapted* and transformed to widely recognized markup semantics.

# The Apache Cocoon Project

## The Design Implementation

In January 1999, we started a software project with the intent of creating a web publishing framework that made it possible to fully separate the work done by people with different skills. The necessity emerged after the problems the author found in creating and maintaining the java.apache.org web site, which grew from being a single software project web site to a container of several software projects under the flag of the Apache Software Foundation.

The web site was designed to reduce work duplication among the different software projects hosted on the site and it was automatically assembled by extracting HTML documentation from the single projects that were stored in a revision control system (CVS).

While this was much more simple and scalable than previous solutions, it became evident that a more radical approach was needed, especially in order to allow the content that made up the documentation to be published in different formats, depending on the user necessities, and styled with different graphical hints to make the publishing more project-specific and portable to other uses.

The hint to the implementation emerged with the release of the first working draft of the *eXtensible Stylesheet Language* (XSL) a few months before. While the language was being designed for client-side use, it became immediately evident to us that doing all the required transformations on the server side would have been much easier than awaiting for an XSL-capable web browser to appear and awaiting for a critical mass of people to adopt it.

Moreover, the availability of both a general markup syntax (XML) and a transformation language (at that time still part of the XSL specification, later moved out into a different specification that became XSLT) made it possible to obtain the long awaited complete separation between content and style with the use of technologies that, once finalized and recommended by the Web Consortium (W3C), would have automatically become standard web technologies.

Cocoon 1.0 was, in fact, just a java servlet that allowed XSLT to be performed on the server side and thus made it possible to *adapt* old and XML-unaware web clients to new markup languages. As a first example, we showed how the rather complex main page of java.apache.org (shown in the picture below), could be marked up with a specific and much simpler language, then transformed to both a complex and fancy graphical version and a simpler version for text-based web clients.

The HTML description of a news was

```
<tr><td bgcolor="#E0E0E0">
<a href="http://java.apache.org/"><strong>Apache JServ User mail
list reaches 500 subscribers!</strong></a></td></tr>
<tr><td bgcolor="#ffffff"><strong>8 Mar 1999</strong> -
The Apache JServ User mail list has offically reached 500 people
subscribed. We would like to thank all of you for being around and
for participating in the making of such interesting projects. This
incredible amount of brain power is what makes us really proud,
and we are looking forward to host more projects and more people
to make what open source is all about: learning, meeting people,
challenging ourselves and have fun!
</td></tr>
```

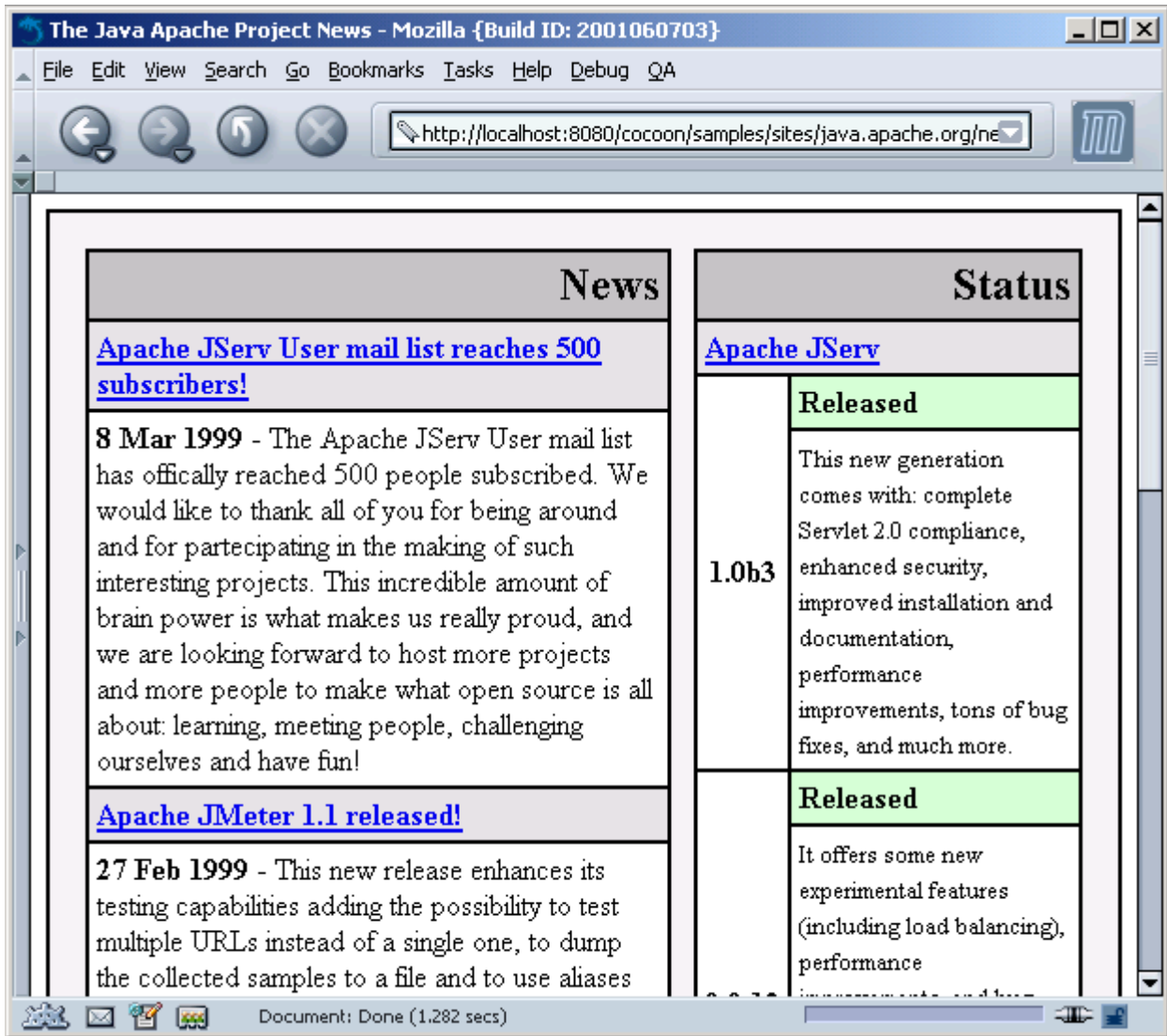
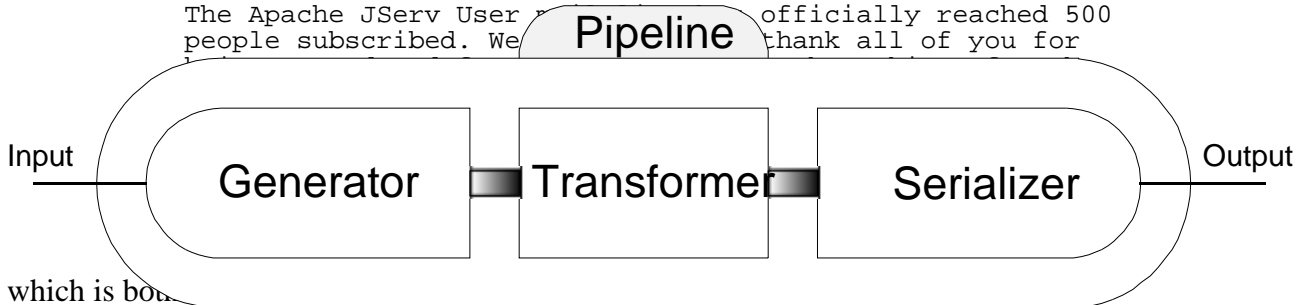


Figure 12 – java.apache.org Main Page

the new XML description with a custom semantic became

```

<news>
  <title>
    Apache JServ User mail list reaches 500 subscribers!
  </title>
  <link>http://java.apache.org/</link>
  <date>8 Mar 1999</date>
  <content>
    The Apache JServ User mail list officially reached 500
    people subscribed. We thank all of you for
  </content>
</news>
    
```



which is both

Figure 13 – Cocoon Pipeline

The concept was very appealing and in March 1999 the Cocoon Project was released to the public as an open source software (thus, freely available along with its source code) and became an

official project of the Apache Software Foundation with the following description:

Cocoon is a 100% pure Java publishing framework that relies on new W3C technologies (such as XML and XSL) to provide web content.

The Cocoon project aims to change the way web information is created, rendered and delivered. This new paradigm is based on fact that document content, style and logic are often created by different individuals or working groups.

Cocoon aims to a complete separation of the three layers, allowing the three layers to be independently designed, created and managed, reducing management overhead, increasing work reuse and reducing time to market.

The Apache Cocoon Project was later moved under the more general Apache XML Project and is now hosted at <http://xml.apache.org/cocoon/>

## Main Architectural Concepts

Cocoon is based around the concept of *resource pipelines*. Each and every resource handled by Cocoon is mapped to a processing pipeline that is responsible for creating it.

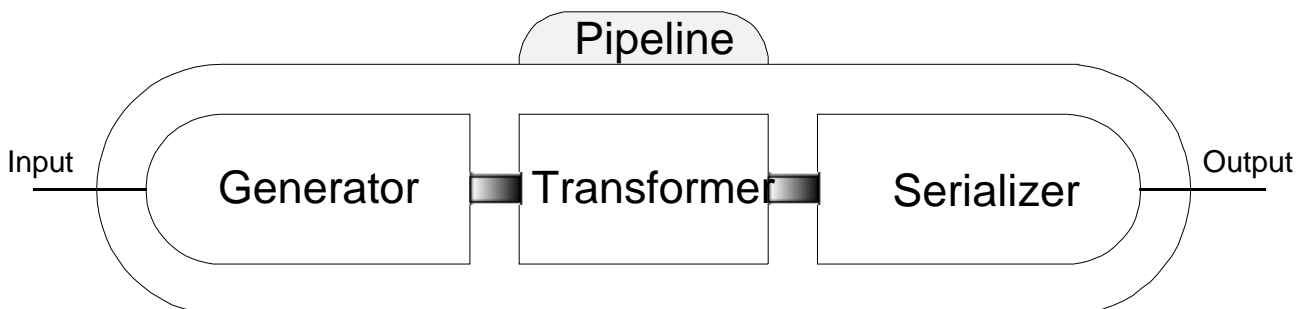


Figure 13 – Cocoon Pipeline

The pipeline is made of *pipeline components* which can be of three types:

- **generators:** are the components responsible for *initiating* the pipeline activity.
- **transformers:** are the components responsible for *transforming* the result of previous components.
- **serializers:** are the components responsible for *terminating* the pipeline activity and prepare the resource for client consumption.

The pipeline *must* have at least one generator and one serializer, but may have any number (from zero to n) of transformers.

This model implements the *pipes & filters* design pattern [Gam95] and resembles the processing model of the UNIX operating system where the output of one process can be redirected as the input of another process.

Just like it happens for UNIX, the ability to modularize processing by connecting pipeline-aware components makes it possible to obtain a wide range of functionality with the simple creation of pipelines with standard components.

Moreover, this modularity allows to extend functionality by creating new pipeline components and still make it possible to connect them to processing functionality already available.

However, there is a substantial difference between the pipeline model in UNIX and in Cocoon, and this is given by the nature of the data sent between the components. While UNIX sends streams of bytes, Cocoon sends streams of events.

## ***Event Based Processing***

Cocoon was originally designed around the concept of creating memory representation of the processed documents and pass this memory model between the components responsible for its processing.

This model presents one major advantage: programming simplicity. In fact, passing data structures between components is a natural way of building a software system and it's much more natural for programmers, especially in the Java world where *object models* are normally used for describing complex data.

Unfortunately, this solution has two major drawbacks:

- *high memory use*: every time a document was processed by a component, a new data structure was required and stored in memory. As the number of components involved in the resource processing and the number of concurrent requests handled by the publishing engine grew, it became evident that such a system would not have scaled as needed.
- *high latency (thus low perceived performance)*: if a memory representation must be build between each processing step, the time taken for the resource to start appearing at the client side will be dependent on the size of the document and not only from the calculation complexity of the processing steps required to process it.

These two issues, alone, made the entire system incapable of matching the serving capacities required by those big and busy sites that mostly required a publishing system.

It was later decided to adopt the architectural solution of event driven processing: instead of creating memory representations and pass them along once completed, each time a piece of a resource was processed, an event would be fired to the next processing stage. This allows, in theory, to deliver to the user some content with a latency time depending only on the processing complexity triggered by that event and not from the complexity of the resource itself.

This radically reduces the latency time on the client side and makes the user perceive a much faster response, improving its browsing experience.

Also, the event can be consumed when processed by the last stage and avoid taking up system memory.

Even if this is not true if the event processing is not stateless and requires information on the location of the event inside the document, it is entirely possible (and this is what was implemented) to design an *event buffer* that can grow and shrink depending on the needs of the logic processing the event.

Needless to say, such a system is much more complex to program and understand than the memory based one, but removes all the performance scalability problems of the previous architecture.

This event-based processing model is defined by the Simple API for XML [SAX2] which is an event-based *application programming interface* that standardizes what events may be fired during XML processing and how they should be forwarded to the application responsible for consuming them.

This said, there are two possible ways of sending data between filters in a pipe (bytes and SAX events) and depending on this there are four possible ways of mixing them:

1. bytes in input, SAX events for output: this is what yields a *generator*.
2. SAX events for input, SAX events for output: this is what yields a *transformer*.
3. SAX events for input, bytes for output: this is what yields a *serializer*.
4. bytes in input, bytes for output: this is what yields a *pipeline*.

It is important to note that any SAX-enabled pipeline can be substituted by a regular byte-oriented pipeline, but it requires a stage of parsing/serializing (marshalling/demmarshalling of XML-structured data) for each component, which results inherently less performant.

## **Centralized Resource Management**

Cocoon produces the requested resource by dispatching the request to the appropriate pipeline responsible for processing it.

In Cocoon's terminology, the collection of the mapping information is called a *sitemap*: a sitemap is a high level description of how the resources served by Cocoon should be processed, together with the information on how to create the processing pipeline.

Cocoon's sitemaps are described as XML documents, using a specific markup designed and maintained internally by the Cocoon Project.

The sitemap represents the central point of resource management because it is possible to acquire a complete description of the resources that make up the site and how they are produced (hence the name *sitemap*).

The sitemap is inherently declarative, follows closely the programmatic model of XSLT, but it was designed around a solid component model to allow most of its behavior to be completely extensible.

Besides the three pipeline components, the sitemap model adds four other components types:

- matchers
- selectors
- readers
- actions

### **Matchers and Selectors**

Matchers are sitemap components responsible for intercepting an incoming request depending on some of its properties. While in XSLT there is only one time of matching and it's hardwired around XPath, the sitemap leaves to the administrator the ability to use (or even write) its own matcher that may well react on more information than simply the requested URI.

Selectors are responsible for selecting between different choices depending on some request property.

Together, these components allow the creation of very complex serving behavior, without requiring any direct programming; for example, a number of complex matching may be imagined:

- reacting on the User-agent parameter in the HTTP request allows the publishing engine to send agent-specific information. This is mostly useful on clients where small implementation differences on the presentation language interpretation require small adjustments. This is mostly evident on WAP environments where several WML browser implementations are available on the market and it becomes tricky to obtain the exact same results with the same document. With

the ability of matching the specific device, the presentation can be tuned.

- reacting on parameters in the HTTP request allows transparent content negotiation. This may include the natural language used, the size/type of the graphics, the compression of the response, etc..
- reacting on time of the day might allow time–dependent processing.
- reacting on server load or available bandwidth might allow the creation of negative feedback to stabilize the system: if bandwidth is low, redirect to the low–resolution stylesheets in order to reduce bandwidth usage. All of this, completely automatic and transparent to the user.

## Readers

Even in XML–centric environments, not all information is written using an XML syntax. Readers provide an old–style byte–input/byte–output component that is used to *read* information directly to the client.

This is mostly used on static and non–XML resources such as raster images, multimedia files (audio, video), VRML files, Flash–encoded graphics, text files, etc.

## Actions

Action are behavioral component that are able to modify the underlying application model and are meant as extensions to the sitemap functionality. They get access to the object model containing all objects that make up a request and make available to the other pipeline components the result of the processing logic included in the action.

# Contract Analysis

## Squaring the Organization Circle

We have indicated several times that a concern layout must be matched by an equivalent technological architecture that allows such layout to exist in practice and not only on paper.

We will now analyze the contracts required in the proposed concern layout and identify the technologies that allow to implement such contracts and the technological constraints that enforce them.

### ***The Required Contracts***

A contract is the interface that is responsible for separating two concern islands, yet allow them to work together in a complete parallel way.

Given the proposed layout of four concern islands, there are six possible unordered connections between these groups. They are:

- design–content
- design–style
- design–logic
- content–style
- content–logic
- style–logic

We will now analyze each contract, showing the required interfaces and observing what technology may implement and enforce it.

### **Design–Content**

The design island is responsible for designing the site and demanding the effective implementation of required resources to the other three groups. In order to succeed, the contracts with the other groups must include all the information required for them to operate.

Since content aggregation and high–level publishing features are now hosted at the design level, the only information required in this contract is the URI space (where the information will be published) and the properties of the required content (format, language, length, type, tone, etc...).

In order to provide this information, the design group must design the URI space and make it available to the content group. This might be automatically generated by transforming the sitemap into an internal document with all the unnecessary information to the content group stripped out.

But the most important technological constraint must be the format in which the content group creates content. This contract is defined by the use of a specific markup language; either a proprietary one that must be designed with a collaboration of all the parties involved, or an existing one that fits the required needs (for example, Docbook [DocBook])

Moreover, since no technical skills must be required in the content island, authoring tools capable

of exporting the content in such a format must be available, either acquired from external sources or internally developed.

At the moment of writing, a number of XML-specific WYSIWIG authoring tools are emerging especially to fit this niche and it's easy to expect an increasing market for such tools that will enlarge the offering as well as the quality of the solutions.

Misuse of such contract can lead to two different problems: broken links (hyperlinks are considered content and normally written by the content authors) and unrecognized format.

The former problem might be avoided with the use of link crawling techniques (such a system is already implemented in Cocoon2) that are automatically able to scan a linked URI space and identify and report broken links.

The second problem might be avoided with the extensive use of XML validation technology that may identifies validity mistakes directly at authoring time or at the time of deployment in the publishing system.

### **Design–Style**

Just like content, the style activity requires knowledge about both the URI space and the format of all the information that must be presented. They are responsible for providing the stylesheets, the graphics and all the other presentation information that make it possible to visualize or otherwise present the requested information to the client.

It must be noted that in our concern layout, the style activity must not be confused with the design activity: the style group is responsible for adding artistic value to the presentation but it is not responsible for designing the user experience or the location of aggregated data.

Therefore, along with the information on URI space and data formats to be presented, this activity requires information on the data format that must be produced after the styling takes place, along with all the problems possibly associated with multiple browsing devices incompatibilities and the issues associated in making sure that the designed content is presented as intended to the user on the device the production chooses to support.

In our context, the style activity is divided into the actual creation of the presented information (normally using dummy content or templates) and the creation of stylesheets that transform content in a specified markup language into the styled resource.

Unfortunately, creating transformation stylesheets by hand is a complex tasks and the state of XSLT authoring tools is very limited ad the time of writing. As for content authoring, with the adoption of these technologies we expect the establishment of a new market that will see web authoring tools including XSLT functionality directly.

### **Design–Logic**

The logic activity is responsible for creating and managing all the information that doesn't require direct human editing.

In our model, this involves both the development of specific software components that implement the functionality required by the design activities, as well as the creation of all those solutions that manage the information available for publishing.

Example of these are adaptation of newsfeeds, mail messages, news messages, weather information, stock quotes, sport events results, info on movie shows, etc..

This group creates the software required for the publishing system to function accordingly to what

the design group has decided. For this reason, along with the URI space and the data formats that are required contracts just like all the other activities, another contract is the name of the software components, together with its behavior, its parameters and its capabilities.

The sitemap is the technological solution that allows design and logic to work together independently once the basic components are outlined and fully described between the two groups.

Without a sitemap controlled by the design activity, resource behavior and mapping would be left to the hands of the logic activity because wrongly believed to be a concern of the logic group, mostly because closely linked to the activities of web server maintenance.

This would evidently be a serious mistake because it would remove the previously established contracts about URI space creating an unnecessary contract that points to the logic area which is normally cause of friction due to the feeling of misuse of the human resources involved.

The sitemap creates a centralized location where the design activity can design and control the entire addressing space and all the resources, enforcing complete separation of the other three groups which are directly instructed by the design group and don't need to interact together any longer.

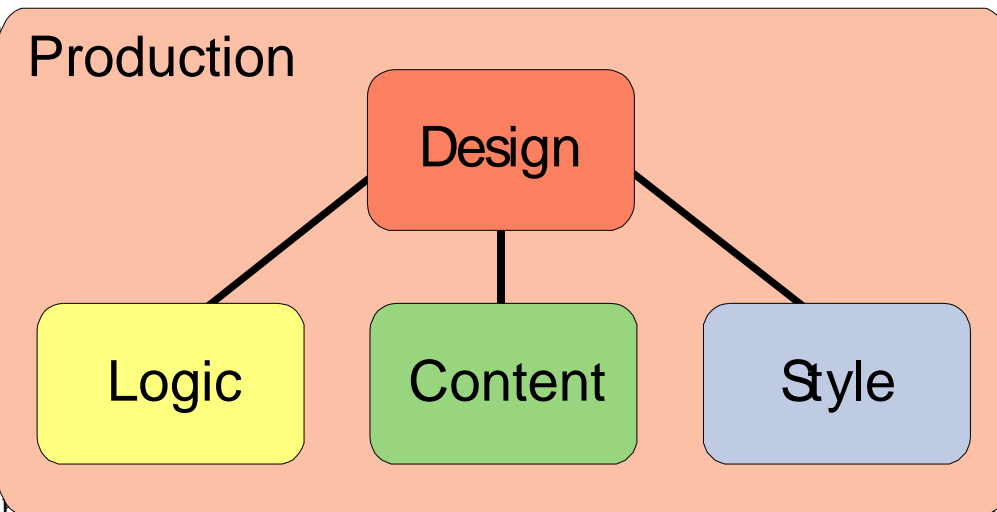
**Content–Style**

The result of the introduction of the sitemap allows the complete removal of such contract because, once all the contracts with the design area are established, these two groups don't need to know the existence one of the other.

This might be the reason of a data format groups.

As an example done in the language.

These examples interaction



or the very reason of a data format groups.

at the work for another

allows all

**Content–**

Even in the they don't

ly because

Figure 14 – Resulting Hierarchical Concern Layout.

**Style–Logic**

The exact same thing can be said for this contract.

**Possible Critiques to This Analysis**

The most important critique that can be moved against this analysis is the fact that the direct interaction between the three underlying groups has not been removed but simply decoupled since all the groups are connected to the design island as a communication hub.

While this might, at first, seem rather artificial and academic, it has a considerable importance: since the design area is responsible for the entire site, it is the only place where a global view of the problematic involved can be found.

Creating direct communications between the underlying groups would *bypass* the area that controls the resources of the site, thus would very likely create ad-hoc solutions to the specific problem, but may later turn out to be a wrong decision, judged with more global information that is not available at the two groups.

At the contrary, forcing all communication and contract establishment thru the design area makes it possible to have a direct control on both the problems and the solutions adopted, as well as a complete vision of what changes need to be reflected on the contracts with the other groups that are affected by the required changes.

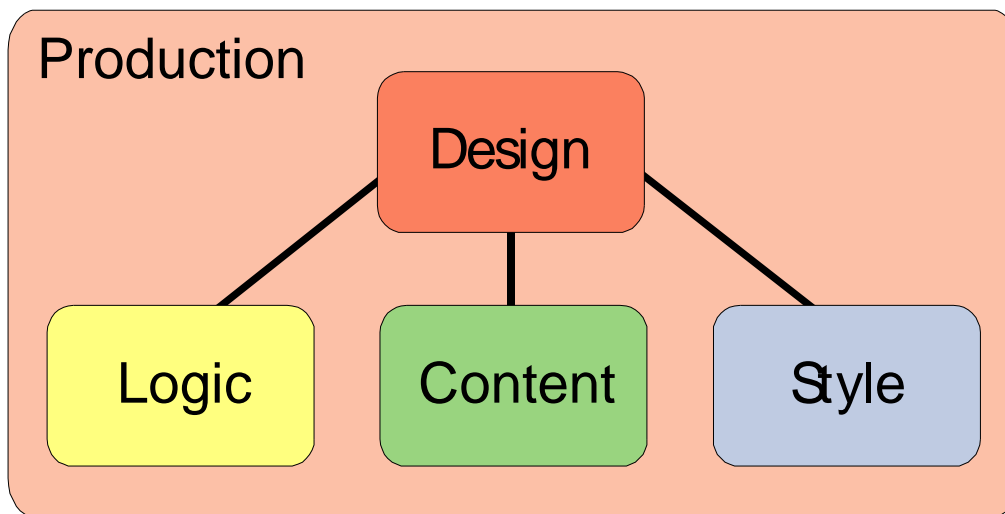
It is also true that such hierarchical concern layout is more subject to organization bottlenecks since the design area is responsible for acting on requests that come from all three groups and dispatch them when they can't be handled autonomously.

It must be noted, however, that the adoption of the *separation of concerns* paradigm allows to drastically reduce the number of communication messages between the concern islands as long as the contracts remain solid.

Great care must be taken in designing these contracts, otherwise, the entire system is very likely to scale *less* than before.

### **Resulting Layout**

After this analysis, the resulting layout is somewhat changed



*Figure 14 – Resulting Hierarchical Concern Layout*

and the *routing* nature of the design island is fully shown.

## Case Studies

In this section, we list two examples that show the industrial recognition that this research has triggered thru the use of the Apache Cocoon publishing framework. The first is an example of use on a public web site, the second is the integration of such XML technologies in one of the key development environments for the enterprise.

### **Operaweb**

[thanks to Dott. Gianugo Rabellino, C.T.O., BiBop Research, Int. S.p.a. for coauthoring this part]

Bibop Research, Int. S.p.a. is an Italian *communication/web agency* whose primary focus is in the integration of different communication strategies such as television, radio, internet and printed media. The ability to interact with different media makes Bibop Research a peculiar company in the communication scenario, the plus being the *cross media* concept itself.

The typical customer of Bibop Research is a large enterprise who asks for an all-round communication plan that needs to be built using the latest technologies and the greatest creativity. In the fast-paced business world of today Bibop Research has to fulfill the customer need to build communication products that are able to react quickly to the sudden changes of today's market.

### **The problem**

Today's Internet publishing is a complex and difficult job, made up from different and somewhat incompatible skills. The HTML publishing model is showing its limits since each and every person in the publishing supply chain has to implement parts of other's jobs and skills.

A quick study revealed that the odds faced by Bibop are a major problem for the whole Web industry. There are many approaches to this but generally they rely on some kind of proprietary technology and custom approach, the usual solution being writers that need to know the HTML language (which they are very uncomfortable at), designers that need to care about inline (logic) code in the HTML source and programmers that waste their skills by writing almost useless HTML and caring about the final aesthetic result.

This model is heavily ineffective and has two major problems with serious impact on the growth of the structure. First of all the resulting site is somewhat "frozen" in its initial state, meaning that content is stored in a format (HTML) that does not allow for easy reuse or restyle of the site itself: as time goes by the situation can only get worse since every document being added to the site will turn out into a problem when refactoring is needed.

The second problem has to do with human resources and is even worse. Every actor in the supply chain has to deal with other's concerns as outlined before: it's easy to imagine that writers can cope with HTML only in a basic way and that their knowledge of the HTML techniques cannot be standardized and is in itself error prone; designers too are not prepared to deal with embedded logic code, and this in turn can lead to inefficiencies and errors; logic programmers do not have any visual orientation and don't deal well with usability and aesthetic issues: this turns out in possible frustrations of art direction's guidelines and loss of user experience in the site.

In the end the whole structure is unable to grow and scale: every addition to the team needs to be trained to work in a highly interconnected structure and has to learn to deal with concerns that are stranger to his own skills and interests. Finally, when a change is needed (or, for that matter, even when the project starts) every actor needs to take part in the project: this means that if the site visual appearance changes even writers and programmers have to be notified and need to participate in the process, which is time and resource consuming.

### **Separation of Concerns**

Since the heart of the problem is the intermixing of skills and concerns it's easy to imagine that the solution is to separate the contexts where people work in order to let every professional do what he does best: in other words the proposed solution was to build an environment based on the Separation of Concerns concept and on the principle of leveraging the skills of every person of the staff.

Those concepts, brought into a web environment, meant splitting the teams that used to work on a site and let them work as separately as possible by using the appropriate tools.

This construction has been taken from the Apache Cocoon publishing framework, a tool based on the Separation of Concern pattern that leverages the latest XML technologies and allows to build strong and independent web teams.

### **Structure**

Adopting a technology such as Cocoon meant a total redesign of the internal structure. There is now a sharp definition of every single task and of the actors of the supply chain. As of now a web project is based on a strong design phase, where the leaders of every team design the requirements and the architecture of the project, define the “storyboard” of the site, decide the XML vocabularies to be used and forecast the steps and milestones of the project itself.

From that moment on every team can work independently, pursuing their own goals. This means, as an example, that once the XML vocabulary has been approved and tested the designers start working on visual appearance and user experience of the site, while programmers write the logic that will generate the content once the site is in production. This also means that even the testing phases are carried out independently, and that the customer can evaluate separately the achievements of the project.

Once the site is in production, finally, concerns are still separated so that bug fixing and new requirements are implemented by the proper team and the proper people: there is no need of technical skills if the site has to change the way it looks, and there is no need of intervention by designers if a technical bug is found or a new feature is requested.

All in all this means great flexibility, independence and faster time to market, together with a great reduction of the “hidden” costs of a project, such as those coming out from the need of a huge interaction between different teams and different people.

### **Case study 1: a “legacy” web site**

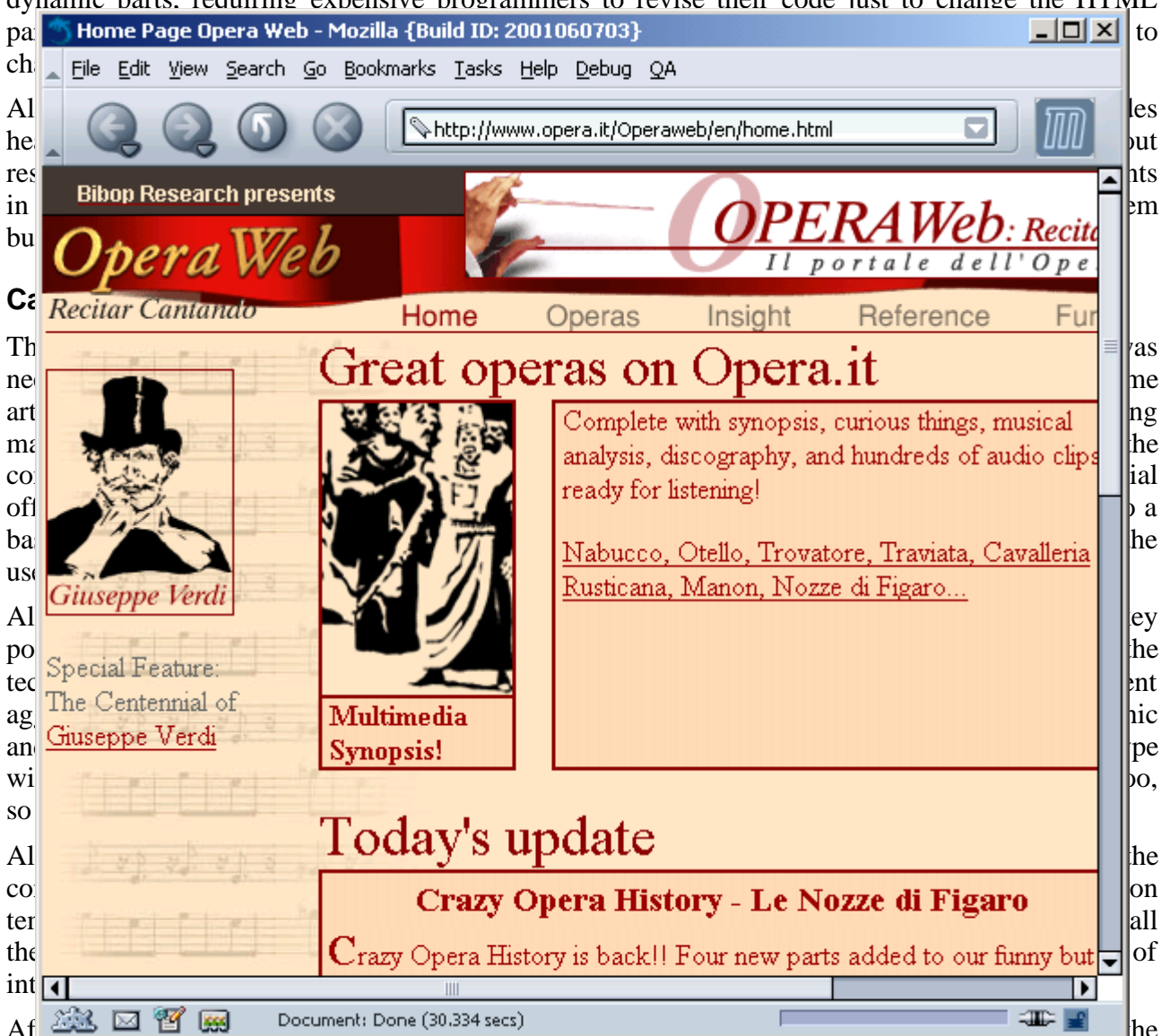
Operaweb (<http://www.opera.it>) is one of the leading sites devoted to the Opera world. The site can be considered a “vertical portal” for the Opera lovers, providing an entry point to the world of Opera on the web. The site is rich of news, articles, singer biographies, discographies, agendas and so on: born in 1996 during the last five years it reached the number of more than 2.000 articles.

The site was designed from its beginnings using a pure HTML approach. Writers used to write content directly in HTML, using tools such as HTML editors. As an alternative for writers who

were unable to use this kind of tools MS Word was used: in this case, though, an automatic conversion to HTML was done, but still it required the intervention of an HTML editor to complete the task including the article text inside the site's HTML. Adding an article also meant manually changing other HTML pages such as indexes and references throughout the site.

The HTML of the site during the years needed revisions not only from a graphical point of view but also due to functional changes. When Operaweb started to do advertising the HTML code for the banners had to be manually inserted inside every single document, and this meant manual intervention of two people for more than two weeks. When a change in the banner management software occurred all the work done previously was useless and needed to be done from scratch: this happened three different times, with major costs to be paid by the management.

The site had interactive sections too, driven by logic code written using the CGI interface initially and PHP later on. This meant that every change on the site had to be propagated also to the dynamic parts, requiring expensive programmers to revise their code just to change the HTML



requirements fulfilled. Since the launch of the site (February 2001) a pool of writers is continuously feeding the data base with articles using a common word processor which saves content using XML, while editors decide about scheduling of the publishing material using an administrative interface that automatically generates and updates indexes and references. Art directors are at work

producing different skins, and they don't need any intervention from the technical department in order to install them on the site. Programmers, finally, do the usual bug fixing and performance tuning with no impact at all on content or visual appearance.

The system is now able to scale. Writers have been added to the team without any training in web techniques. At the moment a total site restyle and redesign is undergoing, again with no impact at all on the existing content (which is stored neutrally as XML) or on the logic used to pull articles from the database: in order to change the site structure only a configuration file editing will be

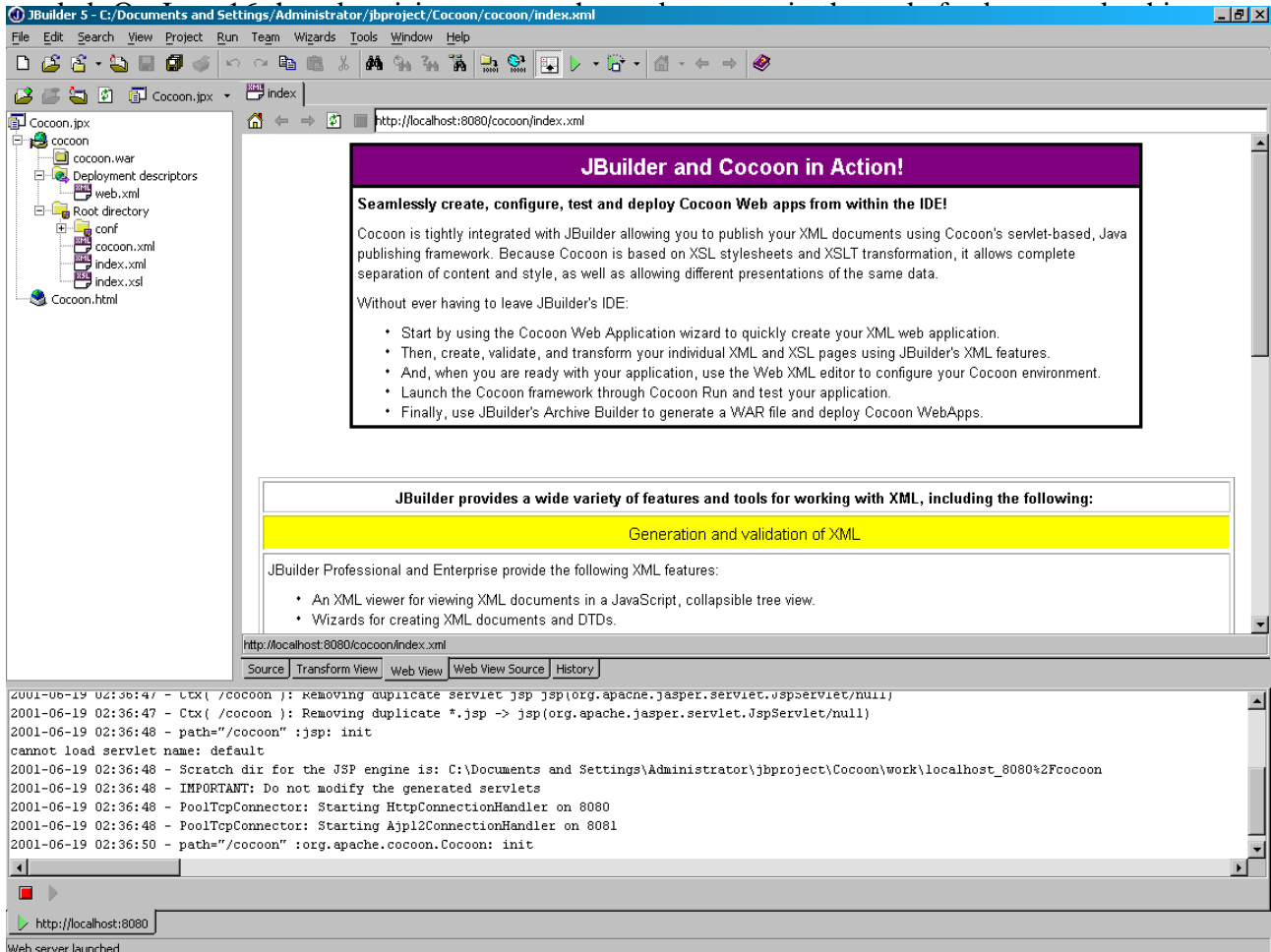


Figure 16 - Cocoon integrated into JBuilder 5

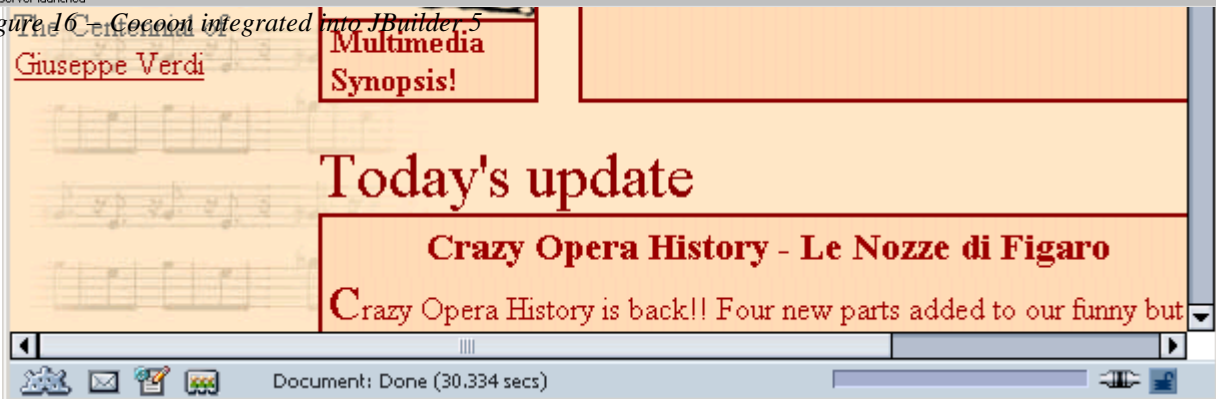


Figure 15 - OperaWeb Front Page

## Borland JBuilder 5

Borland Software Corporation (<http://www.borland.com/>) is one of the leading suppliers of

software development tools and leader in the market for Java *integrated development environments*. Starting from the fifth release, Borland has decided to ship Cocoon in JBuilder [JBuilder5] Professional and Enterprise editions.

Cocoon has been chosen as the de-facto reference implementation of a publishing framework based on XML technologies and the Java language, something that gives big resonance to the project as well as stating the quality of both the design and the implementation.

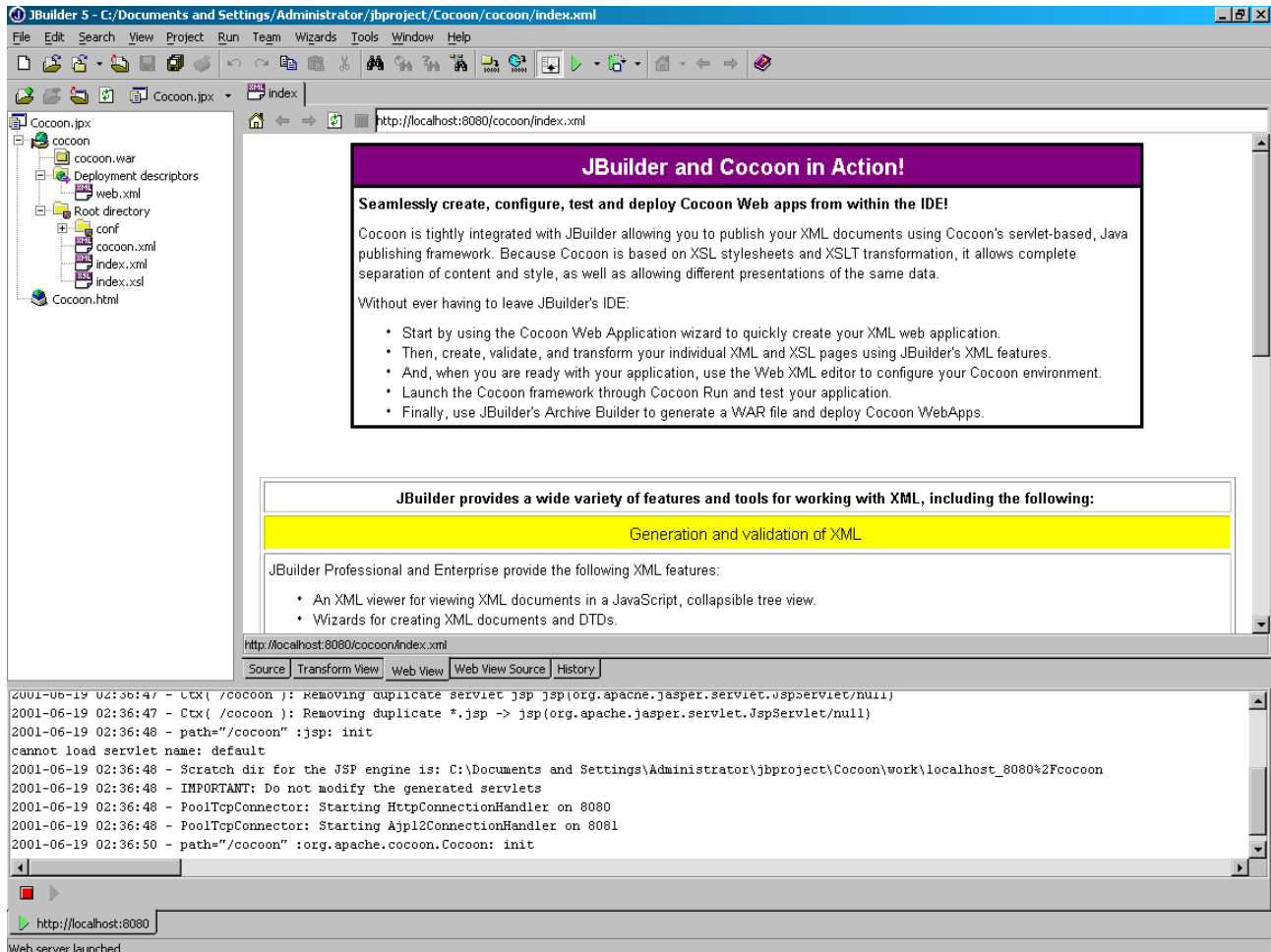


Figure 16 – Cocoon integrated into JBuilder 5

## Other Live Sites

Cocoon is also actively used in several production web sites. Currently, the list of public sites actively powered by Cocoon counts around 50 sites that range from corporate web sites to museum, from news sites to trip planners.

The updated list of public sites powered by Cocoon can be found at <http://xml.apache.org/cocoon/livesites.html>.

## Conclusions

This thesis represents the results of two years of research and development in the field of web publishing. The interest was triggered by the evidence that while existing web technologies allowed the web as a whole to grow with no evident scalability problems, the same could not be said for each site.

In fact, personal experiences showed that with the increase of the people involved in the creation of a web site, the productivity grew much less than linearly and suggested the possibility of a saturating trend that would intrinsically limit the maximum size of the people involved in web production and would have a great economical impact.

In order to show the existence of such trends, we elaborated a simple mathematical model of the work required in web production. Results indicate that such saturation trends exist and are intrinsically tied to the own *fully connected graph* nature of the workgroups involved in web production.

Next, in order to provide a solution that would limit or otherwise reduce the impact of these trends, we observed solutions applied to other fields that exhibited the same issues. The proposed solution is based on the use of *separation of concerns*, a modern software design methodology, applied to the organization of the human resources and their interactions.

Identified a solution, we analyzed the most used organization model and outlined its main problems. We have also shown the importance of having a close match between the technology used and the work organization. In fact, it was observed that separating people in different workgroups doesn't reduce the number of interactions required and doesn't alter the saturation trends, unless a matching technological foundation is introduced.

The effect of growth saturation is reduced only if the number of possible interacting connections between all the people involved is reduced. To make this possible, we propose to separate the human resources in groups that are mapped to concern islands and provide complete separation with the establishing of solid contracts enforced by a technological framework designed around the same model.

Then, we identified the technologies that made it possible to develop such a framework and described the architecture of such implementation that we created in 1999 and evolved since then with the help of the worldwide Cocoon development community.

We finalize the technological analysis along with the matching concern layout by identifying the technologies, solutions or choices that provide and enforce the contracts needed to obtain the complete separation of concerns.

## Future Work

The Web is an extremely successful technology but it's still very young (around 10 years old) and has been advancing so fast that very few technologies in this field can be said mature.

Our work indicates that as soon as the web model is applied to environments that weren't included in its original design, second order effects appear and many times, in order to eliminate them, radical architectural and technological solutions are required.

We firmly believe that we have just scratched the surface of the evolution of web technologies as a far reaching and well established informative system and a lot of work will have to be done both on architectural design on one hand, and social adoption on the other hand.

### **Semantic Web**

The availability of a solid, complete, portable and performant web publishing framework is going to have a strong impact on the evolution of the semantic web [SemanticWeb]. In such a context, the availability of publishing frameworks on the server side might also be very harmful: if the XML technologies are used internally but the semantic content is not directly accessible from the outside, such a site is not, from a global point of view, any different from a site that is hardwired with presentation languages and for this reason doesn't share semantic information.

The semantic web activity focuses on creating technologies that make it possible for the web to be more machine understandable. In order to make this possible, it appears obvious that content must be marked up with information that drives this recognition of what the content *mean* directly rather than how to display it.

In fact, while presentation languages contain information on how to *present* the information to the human user to allow him/her to perform semantic analysis and understand the meaning of the conveyed information, machines have an intrinsically poor capacity of performing automatic semantic analysis, mostly because they lack the collection of experiences that might be identified as *common sense*.

In order for machines to *understand* the information, this must be algorithmically certain and it might be possible, with the use of simple inference rules, to increase considerably the information processing capabilities of the web, which is today limited of serving resources without considering what goes thru the wire.

This said, while it's evident that publishing frameworks could limit the scope of semantically marked-up information, it is also true that they create a strong economic reason for sites to adopt technologies that might otherwise appear academic.

The creation of semantic content is expensive and it's easy to estimate that web sites will not adopt these technologies unless they have a valuable reason to do so, and the reduced production costs associated with the reduction of the impact of growth saturation is very likely to be perceived as such.

Notwithstanding, the semantic web is a global effort and presents the chicken/egg problem: no

semantic-based web services will appear unless there is a *critical mass* of semantic information available on the web, and the costs of creating semantic content will not be paid unless there are semantically-meaningful web services that make use of it and provide a return on the investment.

The adoption of an XML-based publishing framework might remove this stall if semantically-meaningful services are implemented on the local content. An example of this might be advanced local searching capabilities: web site productions might decide to invest time and resources to use fully semantic technologies and add all the necessary metadata information to increase the quality of local searching experience and thus provide a better service to the site users.

The transformation from a human-oriented web to a web where humans and machines cooperate will have to pass thru the cost/benefit analysis of the technology involved and in order to happen, must allow incremental growth starting from local use.

Only when enough semantic content will be written and will be publicly available, such activities will take off and show its effectiveness as a global environment, just like the original web did. We believe that XML-based publishing frameworks will have a major role in making this happening and we'll continue to work in this direction.

### ***Extending The Study on Web Applications***

If the future of publishing goes in the direction of serving machines as well as human users, the future of information systems is heavily web-oriented and many distributed applications based on the client/server paradigm have, or will soon be, ported to web-friendly application environments.

Unlike publishing which is mostly a GET-based activity where resources are requested from the server and consumed by the client without returning data back to the server, web applications normally involve the same problematic of publishing for everything that concerns data presentation and content publication, but have much greater needs in terms of flow control, input validation, user authentication and resource access authorization.

These issues are very likely to give light to architectural limitations of the current work and require modifications of both the architecture and the technological solutions adopted. However, we fully believe that pure publishing and data-driven web applications are the opposite ends of a range of web solutions that spawn all possible weights of the associated concerns.

For this reason, we believe that it's possible to design a single architecture that is capable of providing support for both complex publishing needs as well as complex flow needs, allowing the best of both worlds to coexist. We believe this will be even more important in intranets/extranets than on public web sites. Or in the new potential market that might be created by successful web services.

We will continue to work on the creation of a technological infrastructure that would reduce scalability problems on both web publishing, web application development and all the possible mixes of the two.

### ***Content Management Systems***

Web servers were born as programs that allowed to publish the content of a file system folder. As requirements and functionality grew, the use of file systems to store published information started to appear insufficient.

Features such as revision control, persistence, resource locking and performance limitations became evident and imposed the need for better solutions.

These solutions are called *content management systems* (CMS) and provide all those features that

file systems alone don't provide. Most of the time, they are built on top of database management systems that is used as persistent and transactional storage solution.

Even if it is entirely possible to use an existing CMS for XML content, the intrinsic language-agnostic syntax of XML and its structure cry for an *ad-hoc* solution that would allow not only files to be accessed and controlled (as it happens in today's CMS), but also have a higher granularity and be able to perform queries and extract information from inside the files.

It makes perfect sense, from an architectural point of view, to back up a XML publishing framework with an XML-specific content management system and becomes vital if local searches have to be performed and must interoperate with the internal semantic structure of the markup used to describe the content.

Future work will be needed both in the design of such XML-aware CMS and its integration with publishing systems.

# Appendix A

## Modern Software Engineering Concepts

Designing software is a creative process, for many an art. The act of creating is in the very human nature and can be found in all human activities, but the realm of software design is particular because it removes many of those material constraints that rule other forms of creation.

At first, this lack of constraints boosts creativity and reduces the energy gap required to start, but soon, it turns into a dangerous freedom. Due to time constraints and general complexity, the act of writing software is almost always a collaborative effort. Collaboration takes place only if there is communication and communication takes place when information is exchanged over a common context.

The above might sound as stating the obvious, but being software development rather young compared to other human activities, it generally suffers from the lack of widely acknowledged guidelines that distill wisdom and experience in practices.

Software engineering's main concern is the research and development of such practices, methodologies, guidelines and paradigms that help software engineers to create a common communication context that improves collaboration and offer them distilled experience and mental processes that support the creativity effort.

The ultimate goal is to allow *instant reuse of previous knowledge* without requiring years of try&fail cycles to distill the practices from personal experience. Many believe that software development lacks a solid foundation of practices unlike many other human activities but this a consequence of its youth rather than an endemic problem [Cha92].

This section presents the most important software engineering concepts that were used throughout this thesis. They both provide the language context and a solid cultural background to understand many of the design choices made.

### **Separation of Concerns (SoC)**

The Webster English Dictionary defines *concern* as:

Concern (n.):

1. That which relates or belongs to one.
2. That which affects the welfare or happiness.
3. Interest in, or care for, any person or thing.

Here, we are interested in the third point: a *concern* identifies an area of interest.

The act of separating the concerns identifies the practice of:

1. performing concern analysis on a specific problem/realm given the external constraints.
2. identify the concerns that apply.
3. design the system in such a way that concerns are separated into *islands* that can evolve independently.

Even if the SoC paradigm emerges from programming practices [Cha92][Hür95], the concept is used more broadly in this thesis and is applied directly to human activities. In fact, each individual

has different skills and this automatically translates to different concerns. The use of software design practices that enforce separation of concerns not only improves the software, but improves the quality of the interaction between the software and the users.

For this reason, SoC is also considered a metric for both software development and workflow design: isolating concerns allows to reduce crosstalk and parallelize operation.

## ***Inversion of Control (IoC)***

The first concept that software developers learn is the ability to perform calls to libraries of functions that perform complex services for them. The programmer doesn't need to be aware of the actual implementation of the libraries as far as the API remains the same over time.

Even if this top/down approach allows de-coupling and increases productivity by promoting functionality reuse, it leaves the programmer with the task of designing, developing and maintaining the flow control of the function calls. This is what is simply referred to as *direct control*.

It is common experience that just like functionality, the same parts of flow control are very likely to appear several times across the same project or even across different projects. So, is there a way to promote control reuse?

IoC goes in this very direction: instead of having to rewrite flow control over and over, a basic and general flow control is implemented and customization hooks given to programmers to personalize and modify the behavior for the different project constraints.

Some examples will clarify the situation:

- *OS loader*: all operating systems use IoC when they load and execute applications. When the OS was requested to start an application, this is loaded into memory, the execution environment is setup and the application's entry point is called. The application doesn't need to control directly these operations, thus enforcing separation and portability across different OS implementations.
- *Plug-ins*: a plug-in is a component that performs some additional functionality. This is heavily used for highly modular systems (such as multimedia players, computer graphic applications, etc.) where personalization and extensibility are of maximum importance. A plug-in is a software that gets called by the application using inverted control through a defined interface. IoC greatly simplifies the creation of such plug-ins and enforces portability since the plug-in needs to be aware only of the interface with the external world and not about the internals of the calling program.
- *Interrupt Handler*: it's a function that needs to operate synchronously with hardware event to manage them. Instead of wasting time polling continuously on the resource to listen for the event to happen, the CPU is capable of freezing the state of the system, dispatch control to the interrupt handler and reestablish the previous state at the end of the interrupt routine. The handler program doesn't need to be aware of the operations that are performed by the CPU to save its state, thus simplifying development.

From the above examples emerges the fact that IoC is a very abstract concept and can be applied on almost all situations where flow control can be reused in different situations. This is mostly important on serving environments where much of the flow control is shared across all servers and it's implicit in the client/server model.

## ***Object-Oriented Frameworks***

The use of the Object-Oriented paradigm was aimed at improving software reusability by decomposition, yet provide sufficient evolutionary capabilities with inheritance.

However, the OO paradigm only provides reuse at the level of individual, often small-scale, components that could be used as the building blocks of new applications. The much harder problem of reuse at the level of large components that may make up the larger part of a system, and of which many aspects can be adapted, was not addressed by the object-oriented paradigm in itself.

This led software engineering research to the concept of Object oriented Frameworks which can be defined [Bos97] as:

a set of classes that embodies an abstract design for solutions to a family of related problems.

or, in other words:

a partial design and implementation for an application in a given problem domain.

Since software design and problem analysis occupies much of the initial software development, the ability to reuse design for applications that belong to the same island of concern allows to reduce software implementation costs and allow programmers to focus to specific needs instead of reinventing the wheel every time.

Frameworks are especially effective if combined with SoC and IoC: in fact, a framework can be designed to provide both strong separation of concerns and being the actor for the inverted control, constituting the *skeleton* for the problem-specific components to work together.

Framework development is normally much more difficult than the development of a single application [Lan95][Mat96], this is because frameworks, unlike applications, must apply to an entire set of related problems, thus being much more abstract. Design through abstraction can lead to design mistakes that are very difficult to estimate *a-priori*, especially if no metric was defined for the problem scope.

# Appendix B

## Data Tables for the Growth Saturation Models

The following tables represent the complete data used in the growth saturation models which results are discussed in Chapter 2.

The columns indicate:

- **n** is the number of people involved in the working group
- **Dt** (Direct Time) indicates the time of direct production (in minutes) spent by each individual each day
- **Tot Dt** (Total Direct Time) is total amount of direct time (in minutes) for the entire working group
- **%TT** (Percentage of Total Time) the percentage of time per day spent on direct production by each individual
- **\$/min** (Dollars per Direct Time Minute) is the cost (in dollars) for a minute of direct production activity.

<b>n</b>	<b>Dt</b>	<b>Tot Dt</b>	<b>%TT</b>	<b>\$/min</b>
1	470	470	97.92%	0.21
2	448	896	93.33%	0.45
3	424	1272	88.33%	0.71
4	398	1592	82.92%	1.01
5	370	1850	77.08%	1.35
6	340	2040	70.83%	1.76
7	308	2156	64.17%	2.27
8	274	2192	57.08%	2.92
9	238	2142	49.58%	3.78
10	200	2000	41.67%	5
11	160	1760	33.33%	6.88
12	118	1416	24.58%	10.17
13	74	962	15.42%	17.57
14	28	392	5.83%	50
15	-20	-300	-4.17%	-75
16	-70	-1120	-14.58%	-22.86
17	-122	-2074	-25.42%	-13.93
18	-176	-3168	-36.67%	-10.23
19	-232	-4408	-48.33%	-8.19
20	-290	-5800	-60.42%	-6.9

Table 3 – First Environment

<b><i>n</i></b>	<b><i>Dt</i></b>	<b>Tot Dt</b>	<b>%TT</b>	<b>\$/min</b>
1	470	470	97.92%	0.21
2	457.37	914.75	95.29%	0.44
3	447.67	1343.01	93.26%	0.67
4	439	1756	91.46%	0.91
5	430.81	2154.03	89.75%	1.16
6	422.85	2537.07	88.09%	1.42
7	414.99	2904.93	86.46%	1.69
8	407.17	3257.34	84.83%	1.96
9	399.33	3594	83.19%	2.25
10	391.46	3914.56	81.55%	2.55
11	383.52	4218.68	79.90%	2.87
12	375.5	4506	78.23%	3.2
13	367.4	4776.15	76.54%	3.54
14	359.2	5028.77	74.83%	3.9
15	350.9	5263.51	73.10%	4.27
16	342.5	5480	71.35%	4.67
17	333.99	5677.89	69.58%	5.09
18	325.38	5856.83	67.79%	5.53
19	316.66	6016.46	65.97%	6
20	307.82	6156.46	64.13%	6.5
21	298.88	6276.46	62.27%	7.03
22	289.82	6376.15	60.38%	7.59
23	280.66	6455.19	58.47%	8.19
24	271.39	6513.24	56.54%	8.84
25	262	6550	54.58%	9.54
26	252.51	6565.14	52.61%	10.3
27	242.9	6558.34	50.60%	11.12
28	233.19	6529.3	48.58%	12.01
29	223.37	6477.7	46.54%	12.98
30	213.44	6403.25	44.47%	14.06
31	203.41	6305.64	42.38%	15.24
32	193.27	6184.58	40.26%	16.56
33	183.02	6039.78	38.13%	18.03
34	172.67	5870.93	35.97%	19.69
35	162.22	5677.77	33.80%	21.58
36	151.67	5460	31.60%	23.74
37	141.01	5217.34	29.38%	26.24
38	130.25	4949.52	27.14%	29.17
39	119.39	4656.26	24.87%	32.67
40	108.43	4337.29	22.59%	36.89
41	97.37	3992.35	20.29%	42.11
42	86.22	3621.15	17.96%	48.71
43	74.96	3223.44	15.62%	57.36
44	63.61	2798.96	13.25%	69.17
45	52.17	2347.45	10.87%	86.26
46	40.62	1868.66	8.46%	113.24
47	28.99	1362.32	6.04%	162.15
48	17.25	828.18	3.59%	278.2
49	5.43	266	1.13%	902.63
50	-6.49	-324.47	-1.35%	-770.48
51	-18.5	-943.49	-3.85%	-275.68
52	-30.6	-1591.28	-6.38%	-169.93
53	-42.79	-2268.1	-8.92%	-123.85
54	-55.08	-2974.18	-11.47%	-98.04
55	-67.45	-3709.77	-14.05%	-81.54
56	-79.91	-4475.09	-16.65%	-70.08
57	-92.46	-5270.38	-19.26%	-61.65
58	-105.1	-6095.88	-21.90%	-55.18
59	-117.83	-6951.82	-24.55%	-50.07
60	-130.64	-7838.42	-27.22%	-45.93

Table 4 – Second Environment

<i>n</i>	<i>Dt</i>	Tot <i>Dt</i>	% TT	\$/min
1	430	430	89.58%	0.23
2	400	800	83.33%	0.5
3	386.67	1160	80.56%	0.78
4	377.5	1510	78.65%	1.06
5	370	1850	77.08%	1.35
6	363.33	2180	75.69%	1.65
7	357.14	2500	74.40%	1.96
8	351.25	2810	73.18%	2.28
9	345.56	3110	71.99%	2.6
10	340	3400	70.83%	2.94
11	334.55	3680	69.70%	3.29
12	329.17	3950	68.58%	3.65
13	323.85	4210	67.47%	4.01
14	318.57	4460	66.37%	4.39
15	313.33	4700	65.28%	4.79
16	308.13	4930	64.19%	5.19
17	302.94	5150	63.11%	5.61
18	297.78	5360	62.04%	6.04
19	292.63	5560	60.96%	6.49
20	287.5	5750	59.90%	6.96
21	282.38	5930	58.83%	7.44
22	277.27	6100	57.77%	7.93
23	272.17	6260	56.70%	8.45
24	267.08	6410	55.64%	8.99
25	262	6550	54.58%	9.54
26	256.92	6680	53.53%	10.12
27	251.85	6800	52.47%	10.72
28	246.79	6910	51.41%	11.35
29	241.72	7010	50.36%	12
30	236.67	7100	49.31%	12.68
31	231.61	7180	48.25%	13.38
32	226.56	7250	47.20%	14.12
33	221.52	7310	46.15%	14.9
34	216.47	7360	45.10%	15.71
35	211.43	7400	44.05%	16.55
36	206.39	7430	43.00%	17.44
37	201.35	7450	41.95%	18.38
38	196.32	7460	40.90%	19.36
39	191.28	7460	39.85%	20.39
40	186.25	7450	38.80%	21.48
41	181.22	7430	37.75%	22.62
42	176.19	7400	36.71%	23.84
43	171.16	7360	35.66%	25.12
44	166.14	7310	34.61%	26.48
45	161.11	7250	33.56%	27.93
46	156.09	7180	32.52%	29.47
47	151.06	7100	31.47%	31.11
48	146.04	7010	30.43%	32.87
49	141.02	6910	29.38%	34.75
50	136	6800	28.33%	36.76
51	130.98	6680	27.29%	38.94
52	125.96	6550	26.24%	41.28
53	120.94	6410	25.20%	43.82
54	115.93	6260	24.15%	46.58
55	110.91	6100	23.11%	49.59
56	105.89	5930	22.06%	52.88
57	100.88	5750	21.02%	56.5
58	95.86	5560	19.97%	60.5
59	90.85	5360	18.93%	64.94
60	85.83	5150	17.88%	69.9
61	80.82	4930	16.84%	75.48
62	75.81	4700	15.79%	81.79
63	70.79	4460	14.75%	88.99
64	65.78	4210	13.70%	97.29
65	60.77	3950	12.66%	106.96
66	55.76	3680	11.62%	118.37
67	50.75	3400	10.57%	132.03
68	45.74	3110	9.53%	148.68
69	40.72	2810	8.48%	169.43
70	35.71	2500	7.44%	196
71	30.7	2180	6.40%	231.24
72	25.69	1850	5.35%	280.22
73	20.68	1510	4.31%	352.91
74	15.68	1160	3.27%	472.07
75	10.67	800	2.22%	703.13
76	5.66	430	1.18%	1343.26
77	0.65	50	0.14%	11858
78	-4.36	-340	-0.91%	-1789.41
79	-9.37	-740	-1.95%	-843.38
80	-14.38	-1150	-2.99%	-556.52

Table 5 – Third Environment

# Appendix C

## The Importance of URI Space Design

The URI space of a web site (also known as *addressing space*) is the sum of all the resource identifiers of that site. In general, it can be seen as the equivalent of the directory structure of a file system.

Following the parallel, it's immediate to understand why such addressing structure should be carefully designed, otherwise, as it's evident for file system directory structures, the system becomes easily *messy* and harder to maintain.

Moreover, the URI space establishes the most important contract between the publisher and the user because URIs are the unique address that allows a resource to be accessible and retrievable from all over the internet using the HTTP protocol.

### Good URIs Don't change

Even if the importance of URI space design has been promoted by big names in web design [Bar96][Bar98] and usability [Nie99], many of the issues associated in making sure that a URI can stand the evolution of the resource they identify are normally unknown, not considered or simply ignored.

URI represent the contract between the publisher (producer) and the user (consumer) of the information. Such user may either be a human being or a machine program. In the first case, it is, in general, annoying but not impossible to find a resource that is not anymore identified by the previous known URI: it only requires to go to the home page and use local searching facilities (if available!) of that web site.

A different story is for machine programs which have no way of going around the problem of a *404 Document Not Found* error. This creates the known effect of *broken links*, which are hyperlinks that terminate into URI that can't be mapped to a resource and give the user a bad usability experience and generally a bad feeling about both the pointed resource and the pointing document, even if the latter may not be responsible for it.

Since URIs are contracts, they should be designed as to remain unmodified for years, decades or even centuries.

Even if, in theory, there is no need for web sites to modify their URI space, there are tons of practical reasons to do so. Some examples clarify the issues involved:

### Student's Home Page

Let us suppose that a search on the university search engine yields the following result:

<http://odino.unipv.it/disk2/home/pluto/cool-things.htm>

How *safe* is this URI from changes? How well designed is it?

In fact, the URI is pretty poor and is very likely to stop existing in the future. Let's identify the weak points where environmental changes may effect it.

- *odino* is the name of the machine that hosts the files and it's currently associated with the computer science department in the advanced computer graphics lab. In case the student changes

studies or finishes the courses associated with that lab, the resources associated with that URI could be removed.

- *disk2/home* identifies the name of the disk on which the files are stored. This is another weak point because the files might be rearranged or moved to another disk.
- *pluto* is the user name and might well be reassigned during later years once the student has graduated or moved out of the lab. In this case, the new user might want to use the same URI for something else. This is even worse than a broken link since the user might not notice that the URI owner has changed and the information stored might not even be related to the previous information linked and might completely change the context and/or meaning of the original resource that pointed here.
- *cool-things* indicates something that is inherently time dependent. Tastes and judgment about cool things is very likely to change frequently and force any link to this resource to be time-dependent.
- *.htm* represents the extension of the resource but gives a fault indication that the resource is written using the HTML language, while it might not necessarily be so in the future or if content negotiation is performed by the server depending on the presentation capabilities of the user agent that requests the page.

The parts that are likely to remain safe are:

- *http://* which indicates the protocol used to access the resource and its not likely to change or it's easy to adapt to future uses by providing bridges or adaptors to new protocols yet maintaining this URI prefix.
- *unipv.it* is the domain name and is very likely to remain the same for a long time, at least, as long as the institution that owns that domain survives.

## Dynamic Document Server

Another interesting example is given by the following URI which may be found in a very different context:

<http://www.standards.org/click-wrap.asp?doc=12424323.pdf>

This URI is used to obtain a document from a standard body after reading a click-wrap license that that gives legal information to the user and legal protection to the standard body.

Several problems can be found in this URI:

- *click-wrap* suggests an action that is executed by the logic that controls the publishing of the requested resource. This can be considered a security hazard since it allows people to understand what's going on and allow them to circumvent these actions. Changes in such publishing systems might require changes in the URI space in order to make it more manageable from the technology department but has an impact on the URI space that is unnecessary.
- *.asp* indicates what technology is used to implement this publishing logic and it might well change with time if better technologies are deployed in the publishing systems or simply because the people responsible change and change the technology along with their preferences.
- *?doc=12424323.pdf* is a parameter that is passed to the action implemented by that resource and drives the retrieval process. It presents the same extension problem of the previous example but it's even worse because the URI is now procedural and becomes much harder to remember and understand directly by the users.

Another problem with this addressing scheme is more subtle: supposing the user asks for such a resource and after accepting the license it gets redirected to

<http://www.standards.org/specs/12424323.pdf>

the user will very likely be able to guess the actions taken by the server and bypass the click-wrap scheme by directly accessing another document of which the code is known. For example, by directly accessing:

<http://www.standard.org/specs/4849839.pdf>

it could be able to obtain the specification without passing thru the click-wrap license and without doing anything illegal he could pretend that this spec was accessible with no legal restrictions.

## ***Bad URIs Do Change***

As we have seen from the examples above, a considerable amount of resources must be spent in order to design an addressing scheme that is both simple, easy to remember by the users, secure and capable of standing evolutionary changes of the identified resources.

At least two major side effects can be observed in a hypertext system when the connection between the identifier and its associated resource is modified over time. They are:

- **broken link:** happens when a resource includes an hyperlink to a resource that is no longer available. Broken links are the most visible consequence of a badly designed URI scheme and severely limits the ability of automatic indexers (web crawlers) to create automatic catalogs of the information found on the web. Today, a considerable amount of resources in a search engine are dedicated to finding and removing resources that are no longer available.
- **recontextualized link:** linking is a form of content and provides content contextualization by associating it with a resource that is chosen by the content author in order to provide more information or further clarify the concepts exposed. If a URI changes owner, it is possible that the same URI provides information that is no longer meaningful in the context where it was originally linked. Such recontextualization can be very dangerous because it might be hardly recognized by the user and change the meaning of the resource that includes the hyperlink.

## ***Designing a Good URI Addressing Scheme***

We have said several times that URIs represent the contract between the information publisher and the information user. The existence of URIs made possible the creation of the fully distributed nature of the web as a hypertext system and allowed to remove the bottlenecks associated with centralized systems that couldn't scale linearly with the amount of resources involved.

In fact, URI represent the contract that implements separation of concerns between two or more different publishers, allowing them to work in a complete independence.

This said, it is evident that such independence cannot be complete if one's resource depends on the availability of some other resource that is not tied to a strong and time-independent location. For this reason, the importance of creating a good URI scheme is not different from that of publishing good information or using compatible technologies: a highly unstable addressing scheme will very likely force people *not* to create hyperlinks to your information, thus reducing visibility and limiting publishing power.

A few guidelines can be identified to help in the creation of address spaces. In his style guidelines for the web, Tim Berners-Lee [Bar98] indicates a few things that should be left out of a name:

- **Authors Name.** Authorship can change with new versions. People quit organizations and hand things on.
- **Subject.** It always looks good at the time of creation but changes surprisingly fast.
- **Status.** Directories like "old" and "draft" and so on, not to mention "latest" and "cool" appear all over file systems. Documents change status or there would be no point in producing drafts. The latest version of a document needs a persistent identifier whatever its status is.

- **Access.** Documents that are restricted might later be declassified or otherwise moved into other access limitation schemes.
- **File name extension.** This is a very common one. "cgi", even ".html" is something which will change. You may not be using HTML for that page in 20 years time, but you might want today's links to it to still be valid.
- **Software mechanisms.** Look for "cgi", "exec" and other give-away "look what software we are using" bits in URIs.

Other good guidelines are given by a few questions that should be posed and answered during URI scheme design:

- ***What events might force this URI to change?***  
Answering this question allows to evaluate the events that are likely to happen that might force the people responsible for the maintenance of the URI space to modify the URI or drop it altogether.
- ***How likely are these events?***  
It is impossible to design a URI scheme that can stand every possible environmental changes, but it's entirely possible to estimate the chance of the previously identified events. For example, every student is going to finish its studies, or every specification is very likely to be updated into a new version. At the same time, even if possible, it's unlikely that the domain name of a solid organization changes frequently. With this information is possible to design a scheme that can stand these changes.
- ***Is the URI easy to guess by a user looking for that information?***  
This is rarely considered as a problem, given the availability of bookmark features in almost all browsing agents, but a guessable and fully meaningful addressing scheme would not only improve the user experience and reduce the time it takes to get to a resource, but also reduce server load and the amount of information required by the user to get to the wanted information. Moreover, a clean and meaningful addressing scheme gives a better impression of the internal organization of the published information and creates positiveness in the user.
- ***Should this resource have a URI? does it make sense to bookmark this resource?***  
Likewise, this question is rarely posed and dealt with but it's very important. URI reference resources and nowadays an increasing number of resources are stateful. This means that their result depends on the history of requests made by the user. In these cases, it is completely useless to allow to bookmark or create an hyperlink to a URI that represents part of a stateful resource since when referenced it might yield a completely different result.

A big difference should be made between web applications and published resources since they exhibits different behaviors and require different treatment.

In fact, web applications are normally stateful and should remove the ability for the users to bookmark a resource that is generated out of a stateful application. An example of this is a web mail system where the following URI

<http://www.mysite.org/webmail/inbox?page=3>

gives completely different results depending on the status of the mailbox of the user to which the web application is connected. For this reason, the use of parameters URL-encoding in HTTP GET actions should be avoided in favor of the use of POST actions where parameters passed are hidden and do not make up the URI.

# Appendix D

## Example of a Complex Sitemap

```
<?xml version="1.0"?>
<!-- ===== Cocoon Sitemap Working Draft =====
Copyright (C) 2000 The Apache Software Foundation. All rights reserved.
Redistribution of this document is permitted provided that the following
conditions are met:
1. Redistributions must retain the above copyright notice,
   this list of conditions and the following disclaimer.
2. This document is referred to and considered only as "working draft".
3. Any software implementation inspired by this document must indicate
   in its documentation:
   "inspired by research and development on behalf of the
   Apache Software Foundation"
4. The names "Cocoon" and "Apache Software Foundation" must not be used to
   endorse or promote products inspired from this document without prior
   written permission. For written permission, please contact
   apache@apache.org.
5. Products derived from this document may not be called "Cocoon", nor may
   "Cocoon" nor "Apache" appear in their name, without prior written
   permission of the Apache Software Foundation.
THIS DOCUMENT IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLU-
DING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
This document consists of voluntary contributions made by many individuals
on behalf of the Apache Software Foundation. For more information on the
Apache Software Foundation, please see <http://www.apache.org/>.
=====
This document contains an example used as a working draft for
Cocoon architects to test and understand the issues associated with
sitemaps and XML publishing in general. It must be considered as a working
draft and may be updated at any time.
This document is based on ideas and design patterns inspired by Stefano
Mazzocchi <stefano@apache.org> and Pierpaolo Fumagalli <pier@apache.org>
but grew as a collaborative effort to provide a solid foundation of
design patterns and usability guidelines to the Cocoon Publishing
Framework.
The goal of the sitemap is to allow non-programmers to create web sites
and web applications built from logic components and XML documents.
It finds inspiration from both Apache's httpd.conf/.htaccess files as well
as from Servlet API 2.2 WAR archives. It uses concepts such as Cascading
from W3C CSS, as well as declarative approaches integrated into the W3C
XSLT language. It also uses some element/attribute equivalence patterns
used in W3C RDF.
The following goals were identified as engineering constraints:
- minimal verbosity is of maximum importance.
- the schema should be sufficiently expressive to allow learning by
  examples.
- sitemap authoring should not require assistive tools, but be
  sufficiently future-compatible to allow them.
- sitemaps must scale along with the site and should not impose growth
  limitation to the site as a whole nor limit its administration with size
  increase.
- sitemaps should contain all the information required to Cocoon to
  generate all the requests it receives.
- sitemaps should contain information for both dynamic operation as
```

- well as offline static generation.
- uri mapping should be powerful enough to allow every possible mapping need.
- basic web-serving functionality (redirection, error pages, resource authorisation) should be provided.
- sitemaps should not limit Cocoon's intrinsic modular extensibility.
- resources must be matched with all possible state variables, not only with URI (http parameters, environment variables, server parameters, time, etc...).
- sitemaps should embed the notion of "semantic resources" to be future-compatible with semantic crawling and indexing.
- sitemaps should be flexible enough to allow a complete web site to be built with Cocoon.
- sitemaps should include the notion of "multi-dimensional resource views" even if HTTP doesn't provide them explicitly.
- sitemaps should include the ability to provide resource creation tracing and error handling.

The default namespaces are used mainly for versioning, instead of using attributes such as version="1.0" which could create confusion. People are used to writing URIs with no spelling mistakes, while versioning could be used for their own sitemap versions and this might break operation.

The versioning schema will be "major.minor" where major will be increased by one each time a new release breaks back compatibility, while minor is increased each time a change has been made that doesn't create back incompatible problems.

The syntax

```
<xxx map:value="yyy">
```

is completely equivalent to

```
<xxx>yyy</xxx>
```

throughout the entire sitemap.

```
===== -->
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
<!-- ===== Components ===== -->
<map:components>
  <!--
    Generators generate XML content as SAX events and initialize the
    pipeline processing.
  -->
  <map:generators default="parser">
    <map:generator name="parser"
      src="class:///org.apache.cocoon.generation.FileGenerator" label="content"/>
    <map:generator name="dir"
      src="file:///home/mystuff/java/MyDirGenerator.class" label="content"/>
    <map:generator name="serverpages"
      src="class:///org.apache.cocoon.generation.XSPGenerator" label="content">
      ...
    </map:generator>
  </map:generators>
  <!--
    Transformers transform SAX events in SAX events.
  -->
  <map:transformers default="xslt">
    <map:transformer name="xslt" src="class:///org.apache.cocoon.transformation.XSLTTransformer">
      <compile-stylesheets map:value="true"/>
    </map:transformer>
    <map:transformer name="xinclude"
      src="class:///org.apache.cocoon.transformation.XIncludeTransformer" label="content"/>
    <map:transformer name="schema" src="class:///org.apache.cocoon.transformation.SchemaLoader"/>
    <map:transformer name="rdf" src="class:///org.apache.cocoon.transformation.RDFizer"/>
  </map:transformers>
  <!--
    Readers generate and serialize directly from a resource in binary or char streams for
    final client consumption.
  -->
  <map:readers default="binary">
    <map:reader name="binary" mime-type="image/svg"
      src="class:///org.apache.cocoon.reading.BinaryReader"/>
  </map:readers>
  <!--
    Serializers serialize SAX events in binary or char streams for
    final client consumption.
  -->
  <map:serializers default="html">
    <map:serializer name="html" mime-type="text/html"
      src="class:///org.apache.cocoon.serialization.HTMLSerializer">
      <doctype-public map:value="-//W3C//DTD HTML 4.0 Transitional//EN"/>
      <doctype-system map:value="http://www.w3.org/TR/REC-html40/loose.dtd"/>
      <preserve-space map:value="true"/>
      <encoding map:value="UTF-8"/>
    </map:serializer>
  </map:serializers>
</map:sitemap>
```

```

<indent tab="8">1</indent>
<colors>
  <foreground map:value="white"/>
  <borders>
    <left map:value="blue"/>
    <right>red</right>
  </borders>
  <text>black</text>
  <lines>
    <left map:value="cyan"/>
    <right>orange</right>
  </lines>
  <background>green</background>
</colors>
<map:param name="foo" value="bar"/>
<map:param name="baz">foobar</map:param>
<line-width map:value="120"/>
</map:serializer>

<map:serializer name="wap" mime-type="text/vnd.wap.wml"
  src="class:///org.apache.cocoon.serialization.XMLSerializer">
  <doctype-public>-//WAPFORUM//DTD WML 1.1//EN</doctype-public>
  <doctype-system>http://www.wapforum.org/DTD/wml_1.1.xml</doctype-system>
  <encoding>UTF-8</encoding>
</map:serializer>

<map:serializer name="svg2jpg" mime-type="image/jpeg"
  src="class:///org.apache.cocoon.serialization.SVGSerializer">
  <format map:value="jpg"/>
  <compression-level>80%</compression-level>
</map:serializer>

<map:serializer name="svg2png" mime-type="image/png"
  src="class:///org.apache.cocoon.serialization.SVGSerializer">
  <format>png</format>
  <color-depth map:value="24"/>
</map:serializer>
</map:serializers>

<!--
  Selectors are classes that contain programming logic that perform
  boolean evaluation based on environment state during the call (state
  includes request parameters, machine state as well as any other
  accessible information)

  -->
  Selectors can only respond with true/false when called.
-->
<map:selectors default="browser">
  <map:selector name="load" src="class:///org.apache.cocoon.selection.MachineLoadSelector">
    ...
  </map:selector>

  <map:selector name="user" src="class:///org.apache.cocoon.selection.AuthenticationSelector">
    ...
  </map:selector>

  <map:selector name="ip-filter" src="class:///org.apache.cocoon.selection.IPFilterSelector">
    ...
  </map:selector>

  <map:selector name="browser" factory="org.apache.cocoon.selection.BrowserSelectorFactory">
    ...
  </map:selector>
</map:selectors>

<!--
  Matchers are classes that are able to test if the request parameters
  match the given pattern and, if this is the case, they are able to
  return a Map of tokens that resulted from the matching or any
  depending on the matcher own logic (this is up to the matcher implementation).
-->
<map:matchers default="uri-wildcard">
  <map:matcher name="uri-wildcard" src="org.apache.cocoon.matching.WildcardURIMatcher">
    ...
  </map:matcher>

  <map:matcher name="uri-regexp" factory="org.apache.cocoon.matching.RegexpURIMatcherFactory">
    ...
  </map:matcher>

  <map:matcher name="browser" src="org.apache.cocoon.matching.BrowserMatcher">
    <foo value="bar">baz</foo>
    <lines>
      <left>red</left>
      <right>white</right>
    </lines>
  </map:matcher>
</map:matchers>

<!--
  Action are classes that are able to modify the underlying application model and
  are meant as extensions to the sitemap functionality. They get access to the
  objectModel containing all objects that make up a request. An Action can return
  a Map of tokens that resulted from the processing logic (this is up to the action

```

```

        implementation). An Action is references in the pipeline section with the type
        attribute as it is with all the other sitemap components.
-->
<map:actions>
  <map:action name="session-validator" src="org.apache.cocoon.acting.SessionValidationAction">
    ...
  </map:action>

  <map:action name="db-adder" src="org.apache.cocoon.acting.DBAddingAction">
    <db-connection>postgresql-connection</db-connection>
  </map:action>

  <map:action name="db-modifier" src="org.apache.cocoon.acting.DBModifyingAction">
    <db-connection>postgresql-connection</db-connection>
  </map:action>

  <map:action name="db-delete" src="org.apache.cocoon.acting.DBDeletingAction">
    <db-connection>postgresql-connection</db-connection>
  </map:action>

  <map:action name="form-dispatcher" src="org.apache.cocoon.acting.FormDispatcherAction">
    <layout-description>somewhere/layout-employee-app</layout-description>
  </map:action>
</map:actions>
</map:components>
<!-- ===== Views ===== -->
<!--
  the <view> element introduces the notion of multi-dimensional resource
  views which are an extension to the HTTP paradigm of web resources. Views
  do not increase sitemap functionality, but allow substantial verbosity
  reduction and should help users in the creation of complex sitemaps that
  are able to produce different views of the resource they handle for each
  "aspect" requested.

  Views can be pictured as generator-less pipelines which use, as a generator,
  the result of another pipeline from the "label" they indicate or from the
  position (first/last) in the pipeline.

  Labels can be seen as non-standard exit points from the normal pipelines
  and "first" identifies the position right after the generator while "last
  identifies the position right before the serializer.

  Both generators and transformers are allowed to attach a default label to
  them. If no <label> element is explicitly indicated, the sitemap handler
  will scan for default labels attached to the components, starting from the
  serializer and going backward, until it finds a component to start.
  If none is found, the generator is assumed to be the view generator.
-->
<map:views>

  <map:view name="content" from-position="first">
    <map:serialize type="xml"/>
  </map:view>

  <map:view name="schema" from-label="content">
    <map:transform type="schema"/>
    <map:serialize type="xml"/>
  </map:view>

  <map:view name="semantics" from-label="content">
    <map:transform type="rdf"/>
    <map:serialize type="xml"/>
  </map:view>

  <map:view name="hyperlinks" from-position="last">
    <map:transform src="./stylesheets/xlink-filter.xsl"/>
    <map:serialize type="xml"/>
  </map:view>
</map:views>
<!-- ===== Resources ===== -->
<!--
  the <resource> element is used as a placeholder for pipelines
  that are used several times inside the document. This element
  is redundant and its functionality is not directly related
  to the sitemap, but could be cloned by the use of internal
  XInclude, for example

      <xinclude:include href="#xpointer(resource[@name='Access refused'])"/>

  but given the usability constraints and very specific operation
  it is much easier to include such an element instead of forcing
  the use of xinclude/xpointer.
-->
<map:resources>

  <map:resource name="Access refused">
    <map:generate src="./error-pages/restricted.xml"/>

```

```

    <map:transform src="./stylesheets/general-browser.xml"/>
    <map:serialize status-code="401"/>
</map:resource>

</map:resources>

<!-- ===== Action Sets ===== -->

<!--
the <action-set> element is used as a collection of related actions
that are used either several times inside the pipelines or in conjunction
with form actions which are selected by the sitemap for processing.
These action-sets are referenced in the pipeline section by using a
set attribute instead of a type attribute.
All the tokens returned by the individual actions in a action-set are
collected by the sitemap engine into one big map which can be used as
replacements in src attributes from generators/transformers.
-->
<map:action-sets>

<!--
The following action-set defines:
a) a "session-validator" action which will be executed whenever this
action-set is referenced in a pipeline.
b) several "db-*" action which have an additional action attribute which the
sitemap engine uses as a value to compare against the action value supplied
from the Environment object. Every action with a matching action value will
be processed.
c) a "form-dispatcher" action which will tell the sitemap which is the
next resource to display (usually used for the generator in charge).
-->
<map:action-set name="employee-form">
  <map:act type="session-validator"/>
  <map:act type="db-adder" action="Add"/>
  <map:act type="db-modifier" action="Update"/>
  <map:act type="db-deleter" action="Delete"/>
  <map:act type="form-dispatcher"/>
</map:action-set>

</map:action-sets>

<!-- ===== Pipelines ===== -->

<map:pipelines>
<map:pipeline>

  <!--
  Mount points allow sitemaps to be cascaded and site management
  workload to be parallelized.
  -->
  <map:match pattern="cocoon/*">
    <map:mount uri-prefix="cocoon/{1}" check-reload="yes"
      src="cvs:pserver:anonymous@xml.apache.org://home/cvs/cocoon/xdocs/{1}"/>
  </map:match>

  <map:match pattern="bugs/*">
    <map:mount uri-prefix="bugs/{1}" check-reload="true"
      src="jar://apps/bugs.cocoon#{1}"/>
  </map:match>

  <map:match pattern="dist/*">
    <map:mount uri-prefix="dist/{1}" check-reload="false" src="./dist/{1}"/>
  </map:match>

  <map:match pattern="faq/*">
    <map:mount uri-prefix="faq/{1}" check-reload="no"
      src="jar://apps/faq-o-matic.cocoon#{1}"/>
  </map:match>

  <map:match type="uri-regexp" pattern="^/xerces-(j|c|p)/(.*)$" >
    <map:mount uri-prefix="/xerces-{1}/{2}"
      src="cvs:pserver:anonymous@xml.apache.org://home/cvs/xerces-{1}/xdocs/{2}"/>
  </map:match>

  <map:handle-errors>
    <map:serialize type="html"/>
  </map:handle-errors>

</map:pipeline>

<map:pipeline>

  <!--
  Matchers declarative dispatch the requests to the pipelines that
  match some of their parameters
  -->
  <map:match pattern="cocoon/dist/*">
    <map:select type="ip-filter">
      <map:when test="allowsAddress()">
        <!--
        the <redirect-to> element is used to redirect one requested URI
        to another. This is somewhat equivalent to URI rewriting.
        -->
        <map:redirect-to uri="dist/cocoon/{1}"/>

```

```

</map:when>
<map:otherwise>
  <map:redirect-to resource="Access refused"/>
</map:otherwise>
</map:select>
</map:match>

<!--
  When no "type" attribute is present, the sitemap interpreter will use the
  default one, this allows a very compact and user friendly syntax as the
  one below
-->
<map:match pattern="printer-friendly/*">
  <map:generate src="{1}.xml"/>
  <map:transform src="./stylesheet/printer-friendly.xsl"/>
  <map:serialize/>
</map:match>

<map:match pattern="images/logo">
  <map:select>
    <map:when test="accepts('image/svg')">
      <!--
        the <map:read> element is used to read the src directly without
        applying any processing. This is mostly useful when clients
        are capable of handling XML content directly.
      -->
      <map:read src="./images/logo.svg"/>
    </map:when>
    <map:otherwise>
      <map:generate src="./images/logo.svg"/>
      <map:select>
        <map:when test="accepts('image/png')">
          <map:serialize type="svg2png"/>
        </map:when>
        <map:otherwise>
          <map:serialize type="svg2jpg"/>
        </map:otherwise>
      </map:select>
    </map:otherwise>
  </map:select>
</map:match>

<map:match pattern="restricted/*">
  <map:select type="user">
    <map:when test="is('administrator')">
      <map:generate src="./restricted/{1}"/>
      <map:transform src="./stylesheets/restricted.xsl"/>
      <map:serialize/>
    </map:when>
    <map:otherwise>
      <map:redirect-to resource="Access refused"/>
    </map:otherwise>
  </map:select>
</map:match>

<!--
  Example to show the notion of pipeline labels for view generation.
-->
<map:match pattern="labelled/*">
  <map:label name="links">
    <map:label name="content">
      <map:generate src="./slides/{1}"/>
    </map:label>
    <map:transform src="./filters/add-navigation-links.xsl"/>
  </map:label>
  <map:transform src="./stylesheet/slides2html.xsl"/>
  <map:serialize/>
</map:match>

<!--
  Complex example to show how some xpath-like syntax is used to get access
  to the pattern tokens generated by the matchers.
-->
<map:match pattern="nested-matchers/*">
  <map:match type="browser" pattern="name('Mozilla ?\\{.*}')">
    <map:mount uri-prefix="nested-matchers/{1}"
      src="file:///home/www/mozilla-{1}-{2}/{../1}"/>
  </map:match>
</map:match>

<map:match type="uri-regexp" pattern="([0-9]{4})/([0-9]{2})/([0-9]{2})/">
  <!--
    Here we implement the ability to indicate semantic information
    on the processed URI. This is mostly used to avoid to encode
    URI specific information in the XSP since the sitemap maintainer
    is the only one responsible for managing the URI space. This removes
    a URI contract between the XSP writer and the URI space manager,
    moving it to parameter names which normally change less frequently.
  -->
  <map:param name="year" value="{1}"/>
  <map:param name="month" value="{2}"/>
  <map:param name="day" value="{3}"/>

  <map:generate type="serverpages" src="./dailynews.xsp"/>

```

```

<map:transform src="./stylesheet/{1}/news.xml"/>
<map:serialize/>
</map:match>

<!--
  Here we show the use of action-sets. The scenario used is a form
  generated which enables the visitor to add, modify and delete entries
  in an employee database
-->
<map:match pattern="form/employee">
  <map:act set="employee-form">
    <map:generate src="forms/{next-form}.xml"/>
    <map:transform src="forms2html.xml"/>
    <map:serialize/>
  </map:act>
</map:match>

<map:match pattern="*">
  <map:generate src="{1}.xml"/>
  <map:select type="load">
    <map:when test="greaterThen('2.5')">
      <map:transform src="./stylesheet/low-graphics.xml"/>
    </map:when>
    <map:otherwise>
      <map:select>
        <map:when test="is('Mozilla5')">
          <map:transform src="./stylesheet/xul-enabled.xml"/>
        </map:when>
        <map:otherwise>
          <map:transform src="./stylesheet/general-browser.xml"/>
        </map:otherwise>
      </map:select>
    </map:otherwise>
  </map:select>
  <map:serialize/>
</map:match>

<map:handle-errors>
  <map:select>
    <map:when test="accepts('text/vnd.wap.wml')">
      <map:transform src="./styles/Pipeline2WML.xml"/>
      <map:serialize type="wap"/>
    </map:when>
    <map:otherwise>
      <map:transform src="./styles/Pipeline2HTML.xml"/>
      <map:serialize/>
    </map:otherwise>
  </map:select>
</map:handle-errors>

</map:pipeline>
</map:pipelines>

</map:sitemap>

<!-- end of file -->

```

# Bibliography

- [**CSS2**] B. Bos, H. Wium Lie, C. Lilley, I. Jacobs, Cascading Style Sheets, Level 2, 1998, <http://www.w3.org/TR/REC-CSS2>
- [**JBuilder5**] Borland, JBuilder 5 – Integrated Development Environment, 2001, <http://www.borland.com/jbuilder/>
- [**MathML2**] D. Carlisle, P. Ion, R. Miner, N. Popplier, Mathematical Markup Language 2.0, 2001, <http://www.w3.org/TR/MathML2>
- [**Servlet/2.3**] D. Coward, Java Servlet API Specification, 2001, <http://java.sun.com/products/servlet/>
- [**Cha92**] D. de Champeaux, D. Lea, P. Faure, The Process of Object–Oriented Design, 1992,
- [**HTML**] D. Raggett, A. Le Hors, I. Jacobs, HyperText Markup Language 4.01 Specification, 1999, <http://www.w3.org/TR/html4>
- [**SAX2**] David Megginson et al., Simple API for XML 2.0, 2000, <http://www.megginson.com/SAX/>
- [**Gam95**] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of Reusable Object–Oriented Software, 1995
- [**SGML**] ISO/IEC IS 8879, Standard Generalized Markup Language, 1986
- [**Bos97**] J. Bosch, P. Molin, M. Mattsson, P. Bengtsson, Object–Oriented Frameworks – Problems & Experiences, 1997,
- [**XSLT**] J. Clark, XSL Transformations (XSLT) Version 1.0, 1999, <http://www.w3.org/TR/xslt>
- [**XPath**] J. Clark, S. DeRose, XML Path Language, 1999, <http://www.w3.org/TR/xpath>
- [**SVG**] J. Ferraiolo, Scalable Vector Graphics, 2000, <http://www.w3.org/TR/SVG>
- [**Java**] J. Gosling, B. Joy, and G. Steele, The Java Language Specification, 1996, <http://java.sun.com/docs/books/jls/>
- [**Nie99**] Jacob Nielsen, URIs as UI, 1999, <http://www.useit.com/alertbox/990321.html>
- [**Cla99**] James Clark, Understanding XML Namespaces, 1999, <http://www.jclark.com/xml/xmlns.htm>
- [**Rit00**] Jordan Ritter, Why Gnutella Can’t Scale, 2000, <http://www.darkridge.com/~jpr5/doc/gnutella.html>
- [**Mat96**] Michael Mattsson, Object–Oriented Frameworks, 1996, <http://www.pt.hk-r.se/~michaelm/>
- [**Lan95**] N. Landin, A. Niklasson, Development of Object–Oriented Frameworks