XML Patterns

María Laura Ponisio[*] and Gustavo Rossi[*]
([*])LIFIA, Facultad de Informática, UNLP
La Plata, Buenos Aires, Argentina
Mail: {lponisio, gustavo }@sol.info.unlp.edu.ar

Acknowledgments

Thanks to our shepherd, Paul Asman for his dedication and willingness to get this work right.

Abstract

In this paper we deal with the problem of getting distributed data onto a Web site. We present four patterns that can be utilized to achieve a successful solution in that endeavor. Each individual pattern is a way of solving part of this general problem.

In these patterns XML proves to be a smart way to achieve the goal. Through examples, we show precise solutions that can be used alone or combined. They can be especially useful when developers need to get data that belong to opaque systems, when the separation of data from processing is a must, and when data have to cross platform boundaries.

The patterns use the power of XML to share data between distributed sources as well as to transform XML data on behalf of the user view.

Keywords: XML; XSL; XSLT; Design patterns; Web design

Introduction

We present four patterns dealing with a general problem, getting distributed data onto a Web site. Each pattern uses XML, the eXtensible Markup Language, to form part of the solution. The XML specification offers a way of organizing data so that the data can be shared. XML makes easy to transfer data between platforms and separates data from data transformation.

The first pattern, XML In Out Tray, solves the problem of getting, processing and showing data. In this pattern, XML holds the place of the in and out trays that a worker uses to receive petitions. This pattern solves the problem of getting and giving data from and to applications where the internal processes are hidden.

The second, External Assistant, adds outsourcing to the process of generating an HTML page from an XML document and XSL stylesheet. With a model that transforms XML data while keeping it completely separated from the processing instructions, External Assistant solves the problem of how to add external computation.

Here XML data on one side and the instructions in the XSL stylesheet on the other feed a transformation process that generates output in the form of an HTML file. In this context, External Assistant explains how to call an external process whose results are incorporated to the output, always keeping high modularity and clear distinction of the responsibilities of each component.

The third, Information Grouping, solves the problem of grouping and presenting XML data in an HTML page. Any opaque application running on any platform can to provide XML input. An XSL stylesheet then takes this and generates HTML. This process resembles what SQL's Group By does on database tables.

Finally we present XML Mediator, a pattern that solves the problem of transferring data between foreign applications by automatically collecting data from those applications on behalf of a working client collector. It uses a process that collects XML data from data providers. This process takes the different feeds and builds a compilation of the data in an XML document, even when foreign applications run on different platforms. Through this pattern, a thin client receives distributed information that it requires. Furthermore, the process of collecting information is transparent to the user (usually a person at a browser, but potentially an automated process) except when he declares what data he requires.

The complete code for the implementation of the examples present in all the patterns can be found in [Lifia xml].

1. XML In Out Tray

Intent

Organize the activity of components involved in the process of getting and showing data.

Motivation

Let's assume we have to get data from a source, following some criteria. We don't need all the data held by the source, but some of it. The criteria by which we will retrieve the data comes from some data entry, for instance a user-filled form submitted through a browser. Once we've collected and processed data, our final task is to present it.

We wish the design to be flexible enough so that two distinct computational components are capable of interchanging data without coming together into a single mass of code. For instance, we wish to get data from a source generated by foreign systems. The foreign system is opaque and generates some output that acts as our input. But in the interchanging process only data are interchanged.

Our goal is to develop connections between components through which data can travel while keeping high cohesion and low coupling.

The naive solution is to have just one component to receive the request, fetch data from the sources, process what it has found, and generate the output. This is messy, and could lead to too many entrances in the data source looking for the data. This in turn makes it difficult to optimize the system and lowers performance.

The naive solution doesn't have a clean data interchange with foreign systems. It does not even have a clean data interchange between internal components. The solution therefore forces the developer to understand the data model of the source, and also increases programming efforts and coupling between the internal components. This design also lacks modularity, increasing coupling. It doesn't use the virtues of abstraction, and so suffers from low flexibility and reusability. As a consequence a small change could affect the whole system.

We need to find a useful 'glue' between application components to allow language independence on one hand and different data structure coexistence on the other.

Solution

We have three computational components and an XML file. The first component (IN) receives input data containing the search criteria.

The second component (WORKER) receives the criteria from the IN component and fetches the needed data from the XML file. So the only requirements are that the WORKER must understand the request and the format (structure) of the XML file. The WORKER then processes the data and adds a business rule if needed. For instance, it could contain a business rule that applies a specific discount to items bought by some client, or write an entry to a log. Finally this WORKER sends the unformatted results to a third component named OUT.

WORKER processes data and in this process can format and transform it. WORKER also is responsible of performing whatever function is needed for business logic. It can record an activity in a log table for ISO conformance, for instance. But once WORKER finds out the results, it transfers them to OUT, and OUT performs the formatting and transformation (in the XSLT sense) of the results.

OUT gets the data and builds the output, formatting, transforming and rendering them. OUT is designed upon the output device, in the same way as IN is designed upon the input device.

The XML files store the data. Since it is XML it doesn't matter how different the foreign system is that generated the data, as long as the data have some basic and known structure.

If we need to receive data from different input devices and output it in different output devices, we could use several IN and OUT components, one for each proper final device (browser, handheld, wireless phone, etc.). Figure 1.1 shows component connections.

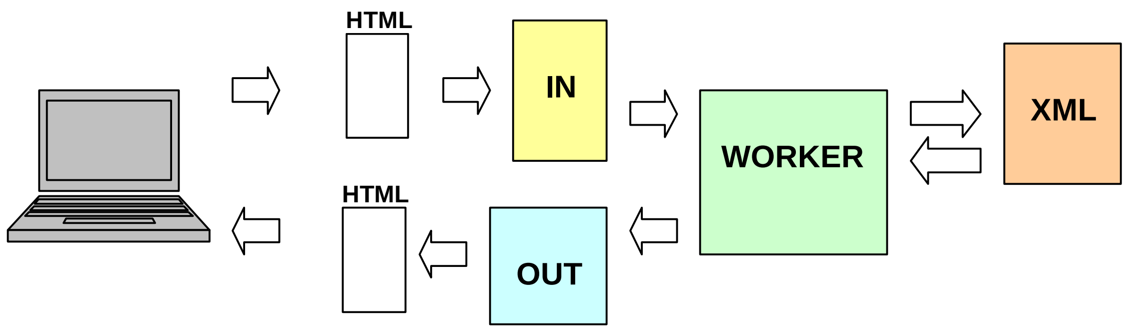


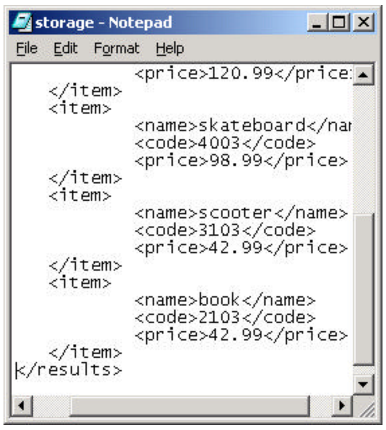
Figure 1.1 Relation between components present in the solution

Example: *Web product catalog display*

This pattern has been used successfully for a catalog of products. We wished to show our products one at a time at customer request. So we built a web application that lets our customers enter some criteria that identify a product, and then goes and fetches the data for the product. In order to fulfill this requirement, we designed a solution based on an XML document that stores our catalog, a component IN that gets the user criteria, a component WORKER that performs the search in our catalog for the right product, and a component OUT that presents it to the customer. Here we included in the patterns only part of the code. Nevertheless the complete code can be seen and downloaded from [Lifia xml].

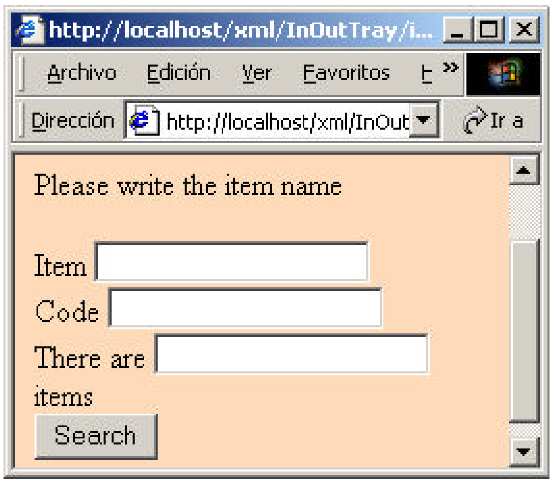
First of all we used the XML document to store the data in our catalog. To do that we defined the structure of our data in terms of XML elements, which together form the XML document. Figure 1.2 shows part of the XML file and the data format we've chosen. Here we have a <item> element for each product and a <name>, <code> and <price> containing information about the item. The list of products present in the catalog is composed of a list of <items>. The XML document is not the ultimate storage of data, but an intermediary between a total different database and our application.

Then we used a component to let the user enter the criteria by which we will search



```
          <price>120.99</price
   </item>
   <item>
          <name>skateboard</nar
          <code>4003</code>
          <price>98.99</price>
   </item>
   <item>
          <name>scooter</name>
          <code>3103</code>
          <price>42.99</price>
   </item>
   <item>
          <name>book</name>
          <code>2103</code>
          <price>42.99</price>
   </item>
</results>
```

Figure 1.2 Part of a catalog in a XML document storage.xml


http://localhost/xml/InOutTray/i...
Archivo Edición Ver Favoritos
Dirección http://localhost/xml/InOut
Ir a

Please write the item name

Item

Code

There are items

Search

Figure 1.3 IN component where user enters search criteria

the product in our catalog. This is accomplished by the IN component, so we built it as a form element in an ASP file, but it could also be a form element in html to get user input. In this example, the IN component is performed by a file called *in.asp*. Figure 1.3 shows the (simple) form that our clients see. Again, the code can be found at [Lifia xml].

At this point we needed something to search the catalog (in fact to search *store.xml*), looking for the product that meets the criteria entered and to present it to the client. So we built *worker.asp*, a program to do the work of the component WORKER and we also built *out.asp*, a program to do the work of the OUT component. Figure 1.4 shows part of the worker.asp's code and Figure 1.5 shows the output presented to the client.

```
'Search xml data from the hints the user entere
Dim xmlDoc
Dim nodeList
Set xmlDoc=createObject("Microsoft.XMLDOM")
Dim str
' Load from a local XML file
xmlDoc.Load "C:\Inetpub\wwwroot\XML\InOutTray\storage.xml"
'Load XML tree node matching user criteria
Set nodeList=
xmlDoc.selectNodes("results/item[name='"strName"']//price")
price=nodeList.item(0).text
listl =nodeList.length

session ("sprice")=price
session ("sname")=strName
session ("sresult") =listl

response.redirect "out.asp"
%>
```

Figure 1.4 A sample of *worker.asp*

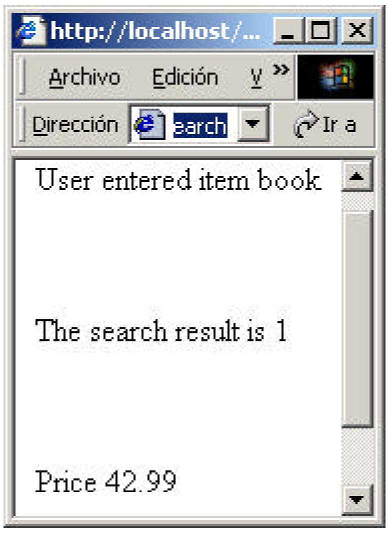
Using the file *worker.asp*, we look for the data we need in the XML document (*storage.xml*) and show it.

Consequences

- We don't get involved with the source data representation.
- Our solution uses abstract components only.
- We can mix the components to better fit the problem context.
- The final design is flexible enough to be used under different circumstances.
- The solution is modular, so it's easier to use, develop and optimize.

Further comments

If we want to keep implementation simple, we can mix components as long as we keep in mind the role that each component plays. But if the activities are really simple or the data are generated within the application (instead of being transferred from another



User entered item book

The search result is 1

Price 42.99

Figure 1.5 Output presented to the client

application), then the use of this pattern is nor recommended.

This pattern wouldn't apply when the components depend heavily on each other in the sense that one component uses code elements that belong to another.

When rendering the data with the OUT component, or after retrieving the data from the XML file with the WORKER, we can group selected data using Information Grouping, a pattern we describe below.

There are no language specific issues. As a consequence, this pattern doesn't restrict the language used for implementation. Furthermore, a key benefit in the design is the use of XML to assure platform independence. This should work even with foreign systems acting as data sources. In the implementation XML acts as a gateway connecting the system that uses this design and neighbours with which the system has to interact.

The XML document plus the programming of the part of WORKER that gets the data from a foreign system provides a placeholder to control access to the data. In this aspect XML In Out Tray resembles the GoF's Proxy pattern [Gamma+95].

2. External Assistant

Intent

Show how to call external computation from an XML-XSL model.

Allow a stylesheet to ask for a task performed by an external assistant.

Motivation

It is a common problem to break a simple architecture to add some kind of computation that performs work needed as part of the output, for instance the task to calculate some value upon some of the data received. This leads to complex and expensive code both in writing and in maintenance, and this in turn ends up raising the cost of the whole system.

So we need to add new functionality but at the same time avoid code complexity. In calculating some value based upon some of the data, for example, we need to figure or fetch some information by some kind of computation that we do not wish to include in the stylesheet, even if the expression relies upon the XML data stored in an XML document. The inclusion could strain the simpler architecture to a breaking point.

If we are working in a context where we need to keep data separate from instructions by using XML and XSLT, adding tasks can increase the complexity of the code. So the designer ends up with one of two naïve solutions: breaking the separation between data and transformation, or forcing the underlying XML+XSLT+parser architecture to handle more that it should.

For instance, we could receive more data than we need to show. We could then select some data and perform certain specific calculations before submitting it for output. The naïve solution would be to overload the stylesheet past breaking.

Solution

Make the stylesheet call Java class methods.
On one side we have an XML document containing data to process. On the other we

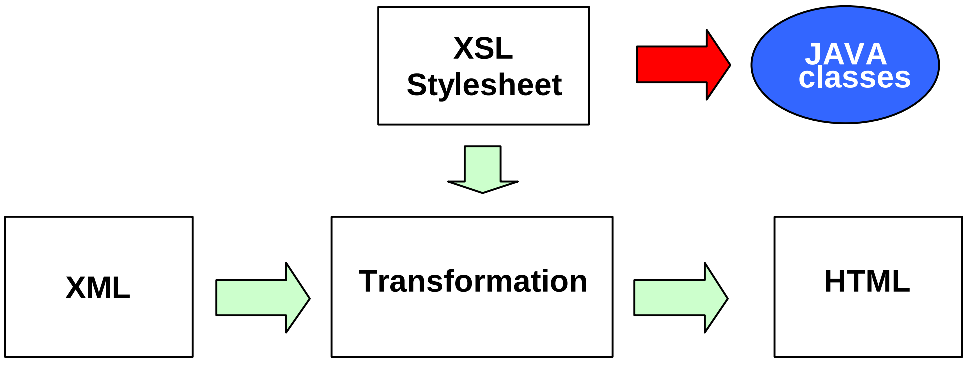


Figure 2.1 Stylesheet calling an external assistant in a transformation process

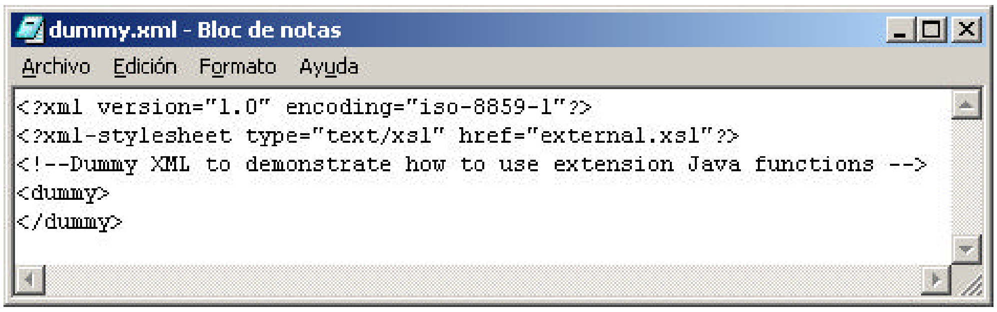
have a XSL stylesheet with precise instructions about what to do with the data. Both of them feed a transformation process that renders complex output. But the process is aided by an external function that adds functionality and then improves the behavior defined in the stylesheet. Figure 2.1 shows the sketch of the solution.

Example 1: *Calling a Java aid from a stylesheet*

In this example we call an external Java function and show how to call a Java class method, which returns the number 1 for demonstration purposes. It symbolizes the call of a program that performs some task.

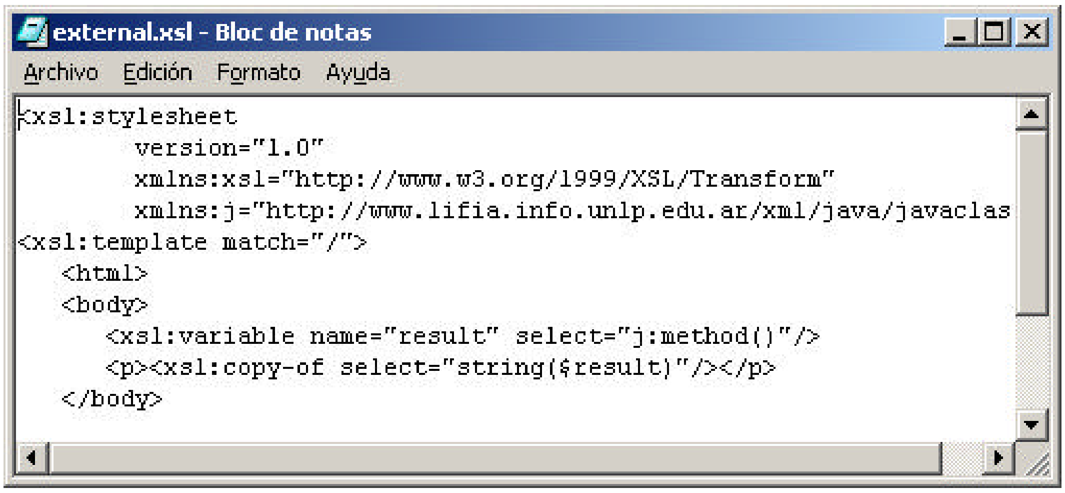
First, we have an XML document performing as a dummy file called *dummy.xml*. In Figure 2.2 we can see the notepad showing *dummy.xml*.

Then we have a stylesheet call a Java class method that returns something (the number 1, in this example). Figure 2.3 shows part of the code of the stylesheet.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="external.xsl"?>
<!--Dummy XML to demonstrate how to use extension Java functions -->
<dummy>
</dummy>
```

Figure 2.2 XML document dummy.xml



```
<xsl:stylesheet
        version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:j="http://www.lifia.info.unlp.edu.ar/xml/java/javaclas
<xsl:template match="/">
   <html>
   <body>
      <xsl:variable name="result" select="j:method()"/>
      <p><xsl:copy-of select="string($result)"/></p>
   </body>
```

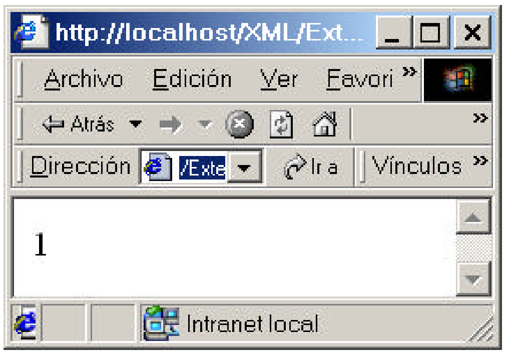
Figure 2.3 Part of the stylesheet external.xsl calling a method

In addition to the XML document and the stylesheet, we need a Java class method to perform the task of returning the number 1. This Java class method is called by the stylesheet and appears in Figure 2.4.

Finally, a parser performs the transformation and leaves the result of its work in an HTML file called *external.html*, which we show in Figure 2.5.

```
public class javaclass
{  public static int method()
   {  int n = 1;
      return n;
   }
}
```

Figure 2.4 Method called by the stylesheet



1

Figure 2.5 HTML file generated by the parser

Example 2: *Inserting current date in a page*

We use this solution when we are building a site and we want to add the current date to a page before submitting it to the user. We accomplish this task by applying our solution together with the Java standard class library.

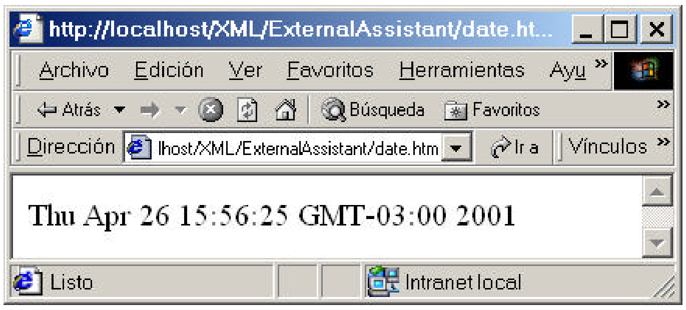
Here we show how to call a method of the Java standard class library. To do that we

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

xmlns:fecha="http://www.lifia.info.unlp.edu.ar/xml/java/java.util.Date"
>
<xsl:template match="/">
   <html>
   <body>
      <p><xsl:value-of select="date:to-string(date:new())"/></p>
   </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

Figure 2.6 *date.xsl* inserting the current date in a page

start with the same *dummy.xml* file used in the previous example. We also have a stylesheet that calls java.util.Date, creates a Date object and initializes it with the current date. This stylesheet is *date.xsl* shown in Figure 2.6. Finally, Figure 2.7 shows the result of running a parser on the XML dummy document, for which the stylesheet date.xsl builds a page


http://localhost/XML/ExternalAssistant/date.ht...
Archivo Edición Ver Favoritos Herramientas Ayu
Atrás Búsqueda Favoritos
Dirección lhost/XML/ExternalAssistant/date.htm Ir a Vínculos
Listo
Intranet local

Thu Apr 26 15:56:25 GMT-03:00 2001

Figure 2.7 Page with the system date generated by the parser

containing the current system date after calling a Java standard class method.

Consequences

- We keep high modularity because we add external computations maintaining the abstraction of the behavior. We avoid mixing external code with the code needed to show the XML data, which is in the stylesheet.
- We lower stylesheet complexity while increasing computation. Today, many parsers allow calling Java class methods. Note how the benefits of this model increase as we need to add computations to the stylesheet to show data processed according to certain criteria.
- This approach also makes it easier to personalize output.
- We have found a simple way to add computation when received data aren't sufficient for what we want to show, and intermediate calculations are needed. [Nielsen99]

Further comments

The mechanism used here isn't defined in XSLT or XPATH specifications. However, some parsers support its implementation. It will probably become a standard in the future [Kay00].

We can use this pattern when the parser that will perform the transformation has a mechanism for binding to external functions written in Java. Since at the time of writing it is not defined in the XSLT or XPath specification, care must be taken, because using this mechanism lowers the portability of the stylesheet considerably. If the developers call a Java class, since it is not still written in the spec, the way they call it depends on the processor. So the fact that a stylesheet works with one XSLT processor doesn't mean that it will work with other. Once that subject is solved, we can use External Assistant with patterns whose solution uses the XML+XSLT+parser model, like the Information Grouping pattern below.

It's a delicate matter to decide whether to use this pattern or Information Grouping. In Information Grouping there's an example of when the decision process reveals that is better not to call an External Assistant. That is because the XSLT spec includes sorting, which is prima facie evidence that this sort of computation should be handled in the stylesheet. But when the process that needs to be performed isn't included in the XSLT specification, then External Assistant can help.

3. Information Grouping

Intent

Model a transformation in XML using XSLT. Given an XML document, the pattern shows the way to format the XML data and generate an HTML file containing the data of the XML document, but grouped by certain criteria.

Motivation

We need to show elements grouped by some criteria, but we have those elements stored without the desired order. In other words, starting from an XML document storing elements, we wish to group the elements on the basis of some subelement. Therefore we build a way of presenting (showing) the resulting ordered list. We use a stylesheet to generate the transformation that we want. For instance, we could build a web mail application to let web clients to get their mail. While doing this, we should offer our user the choice of how to see his mail messages. He should decide if he wants to see the list of messages grouped by date, sender or subject, in ascending or descending order.

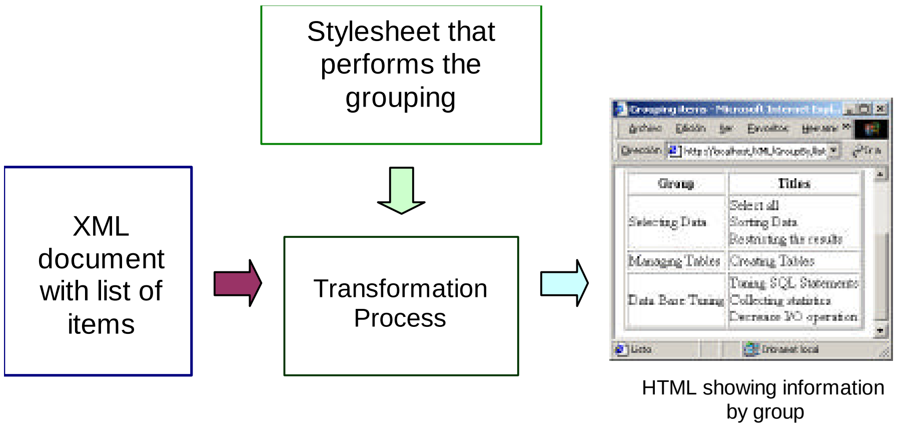
Furthermore, the approach followed in this pattern is useful when we need to keep the presentation logic separated from the business logic. See Further comments in XML In Out Tray and Further comments and Consequences in External Assistant.

Solution

Have a stylesheet group the elements. We have an XML file with the data, and we have to create a file with the transformation. This file is the stylesheet. The stylesheet instructs a program (a parser) what to do with the data stored in the XML file. It specifies a transformation (in the XML sense, since data in the original file remains the same) that traverses the XML tree implicit in the XML file that has the data, and makes a new list which has all the groups existing in the tree. The groups in this list are distinct.

In a second step, the transformation loops through the distinct group list generated previously in such a way that for each group in this list, it inserts a row in a table that will be shown in the output. Then it walks again through the complete XML document tree collecting all the elements belonging to the current group and showing them.

It renders the desired part of each element in the corresponding element of the HTML file. For instance it prints part of the element in the second column of the corresponding row of a table. The group to which the element belongs is the deciding factor of the proper row. As a result the parser builds an HTML file. Figure 3.1 shows the basic solution where an XML file and a stylesheet feed a transformation process that produces an HTML page containing data properly grouped.


Stylesheet that performs the grouping
XML document with list of items
Transformation Process
Grouping Items - Microsoft Internet Expl...
Archivo Edición Ver Favoritos Herramientas
Dirección http://localhost/XML/GroupBy.list
Ir
Listo
Intranet local
HTML showing information by group

Group	Titles
Selecting Data	Select all Sorting Data Restricting the results
Managing Tables	Creating Tables
Data Base Tuning	Tuning SQL Statements Collecting statistics Decrease I/O operation

Figure 3.1 Transformation process

Example: *Retrieving information from a knowledge recorder*

In this example we show how to group XML data according to some criteria and then present it to a client in a browser with an HTML page. The items we work with represent knowledge acquired on a subject. So the list of items is a small store of knowledge kept in an XML document. This list of items is made of subject elements. So our task is to get an XML document containing a list of subjects with no particular order, group those subjects and present them, grouped, on a table in an HTML page. Each subject has its specific format (structure) and belongs to a group in such a way that all the subjects can be placed under some group they belong to. Figure 3.2 shows part of *list.xml*, the document that stores our knowledge data.

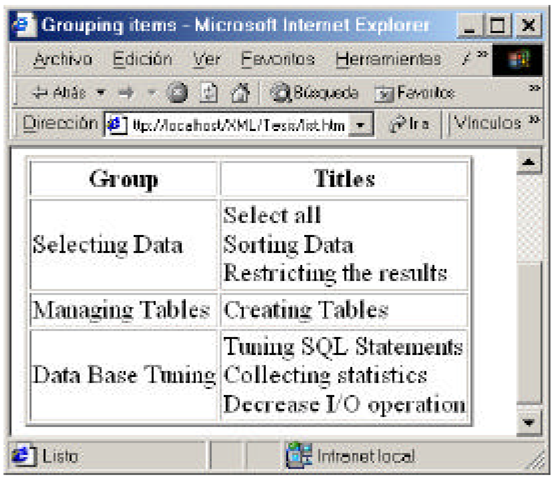
```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="list.xsl"?>
<list>
      <subject>
              <title> Select all </title>
              <group>Selecting Data</group>
            <description>This   is   the   common   problem   o
selecting every item in a table</description>
            <solution>
                  ...
            </solution>
      </subject>
      <subject>
            <title> Creating Tables </title>
              <group>Managing Tables</group>
            <description>We need to create a Data Base table
              </description>
            <solution>...
            </solution>
      </subject>
      <subject>
              <title> Tuning SQL Statements</title>
              <group>Data Base Tuning</group>
              <description>How  to  write  a  SQL  statement  t
minimize data retrieval cost. </description>
              <solution>Use indexes</solution>
      </subject>
      <subject>
```

Figure 3.2 Part of list.xml

Now we need to work with the data so we can transform the XML tree to get another XML tree with the same data, but ordered as we want to show it. So we create a stylesheet that tells the parser to produce an HTML file showing the data in conformance with our specification. In this example we output the title and the group to which the subject belongs (the Owner Group). The output is shown in a browser conforming the rule that titles must be gathered by Owner Group. Figure 3.3 shows a sample stylesheet, *list.xsl*, the stylesheet that we used. Finally in Figure 3.4 we present the parser generated output as a user sees it in a browser.

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
<xsl:template match = "/">
<html>
<head>
   <title>Grouping items</title>
</head>
<xsl:variable name="distinct-groups"
      select="/list/subject[not(group=preceding-
sibling::subject/group)]/group"
/>
<body>
   <h1>Group by</h1>
   <h2>Titles gathered by 'group'</h2>
   <table border="2">
      <tr>
        <th>Group</th><th>Titles</th>
        </tr>
<xsl:for-each select = "$distinct-groups">
           <tr>
```

Figure 3.3 *list.xsl*, the sample stylesheet


Grouping items - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas
Atrás
Búsqueda
Favoritos
Dirección
ttp://localhost/XML/Tesis/list.htm
Ir a
Vínculos
Listo
Intranet local

Group	Titles
Selecting Data	Select all Sorting Data Restricting the results
Managing Tables	Creating Tables
Data Base Tuning	Tuning SQL Statements Collecting statistics Decrease I/O operation

Figure 3.4 HTML showing data grouped by the value of the <group> element

Consequences

- There's a clear gap between the data content and representation format. This advantage is forced by XML. The content-representation separation leads to easy-to-try prototypes. For instance, we can test the transformation while not knowing the actual data yet. Besides, a change in the transformation doesn't affect data. While keeping consistency between XML document and tags used in the transformation, a change in one doesn't affect the other.
- Reusability. This transformation can be used whenever we need to group items by criteria shown in one of their elements or attributes. Note that there are some restrictions to the file storing the data, for instance that the data must be an XML document. But this fact leads to an increase in the number of contexts (situations) where we can apply Information Grouping with low effort.
- It solves the problem of showing a list of interleaving items.

Further comments

Information Grouping can organize data in a different structure (format) from that in which it is stored. For instance if the XML document that stores the data have elements with attributes, then this pattern can be used to group by attributes as well.

This pattern would prove useful grouping an XML format storing mail by sender, for instance.

Information Grouping should be called if the calculation we want to apply to the data is included in the XSLT specification. This stands in contrast with External Assistant, which should be called when the data processing includes some specific calculation that is not part in the specification or, even if specified, will require code that is hard to write.

Nevertheless, when Information Grouping is used, it could be used with External Assistant to add processing without breaking the architecture of the Information Grouping's approach.

4. XML Mediator

Intent

Organize XML information gathering from a wide range of sources.

Motivation

XML is a good, extensible way to collect information from disparate sources. This pattern takes advantage of that to answer the problems of manual data integration, verification of transmission completeness and transparent addition of components.

To collect information from several sources nowadays the user is often forced first to look for the information and then to copy and paste it to some kind of notepad, learn it by heart or print it. He must then repeat the process until he gathers all the information he needs. Note that the information could be spread over several sites. For example, consider a user planning his or her vacations. He must visit several different sites in order to gather information about weather, prices, availability of hotel rooms, flights, etc. This is a tedious task and error prone. In fact, we could argue that Internet acts as a mainframe because the users can see pages one at a time, which makes it difficult to catch scattered data. In order to get the wanted data, the user has to scribble them in a notepad.

Here, unrelated applications must share information. We need efficiency in data transference as well as high performance in handling simultaneous requests. Besides, we want to carry information from several applications into one in a comprehensible format.

We want to translate the information to a universal format to carry it through applications so we can use the formatted data generated by a foreign system. The data coming from a foreign system being formatted is text where markup indicates an inner structure, for instance the formatted text can be a list of elements, which also are text. The pattern used in those foreign systems to produce XML doesn’t affect the use of this pattern. The XML data are created by the foreign system or are data from a foreign system where the XML is added outside that system. If the component systems do not provide XML, an additional process must take the data provided by the components and creates XML from it. Besides, we need some inner validation for the data carried, in order to verify that we’ve received all the data that were sent. For instance, if we were carrying an invoice, we would need to guarantee that we’ve received every item of it in the transaction, and that none was left out due to problems in the transference.

In addition, we want the design to be modular so we can increase the abstraction. Sites offering information and sites gathering it should both add the proper abstract functions for giving and taking information.

Furthermore, there’s a need for easy-to-implement solutions for the interacting sites. This is because adding the connection involves adding programming, and the new component shouldn’t interfere with the design of the site. For instance, it shouldn’t affect the navigational layer, except for adding the connection module to let the user download the data. At this layer we just need to add the component used for providing the information.

Solution

Each application that wishes to offer information should add a component from which clients can take formatted information or have an additional process outside of the foreign system to format the data into an XML valid format for the clients to consume that data. This component interacts with a user-guided client component. Both of them couple at the user's request and allow information transference using XML.

We suggest that web sites include a connection device to let the user have the data that belongs to external data sources like foreign, non-related systems. Once the user has found what he's looking for, he can order the download of the data avoiding the task of copying and pasting data.

We call a *server* each site that offers information. The mating component, *information collector*, acts as a client asking for XML formatted information on behalf of a user and belongs to a separate system.

So sites must have a specific object for giving information (the connecting component) in such a way that client mates (information collectors) can get it through interacting with it.

We then build the client application that makes requests to server applications.

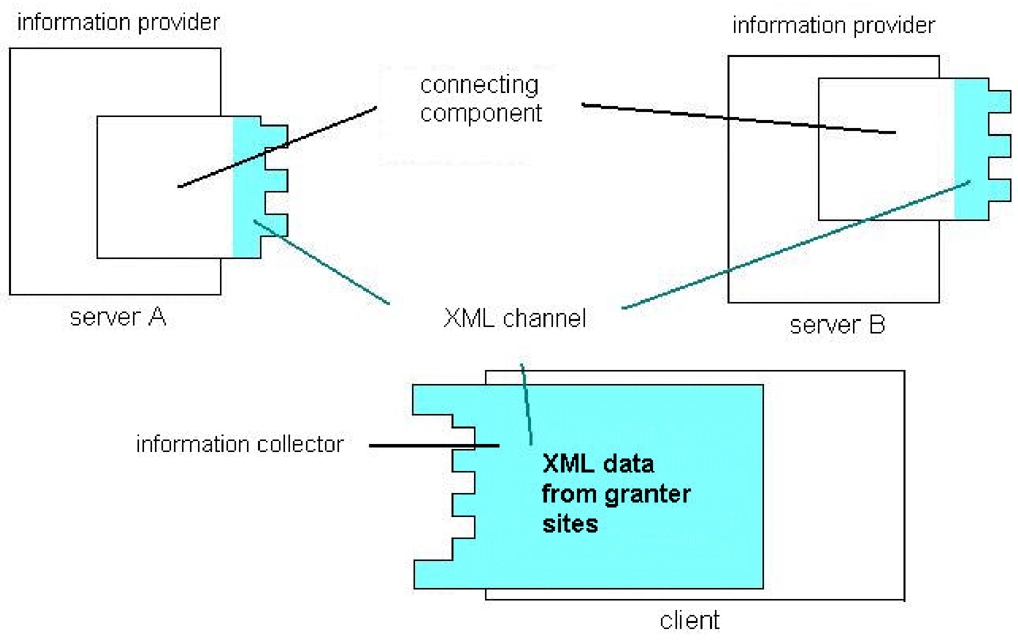


Figure 4.1 Information gathering with XML

Servers connect to the client one at a time.

The information collector component acts under user guidance and plugs into the connecting component, its mating component on the side of the application that's offering information.

Using XML, both kinds of components match perfectly and are capable of transferring information that follow XML standard.

Example: *Getting XML data from foreign web applications and storing it in a XML document*

In this example we used the solution to build a web application that gets information from several different servers and then stores the gathered data for the use of a web client. The sites that offer information are the *servers*. Each server has a component to eventually connect to a client. These components are the *connecting components*.

We also have the *client* that collects information from servers. In order to accomplish it's task, the client has a component that matches and plugs into the server's *connecting components*. This component is the *information collector*.

Once the user has found the information on one of the servers, the client connects its information collector to the server's connecting component and starts the XML data transmission. After receiving this server data, the client connects its information collector to the connecting component of the other server and repeats the process. By doing this, the client creates an *XML data pool* with the information obtained in the web sites. For instance if the client is planning vacations then he goes to the first server to reserve seats on a plane and to the second to book hotel rooms. The XML data pool is in fact an XML document. Figure 4.1 presents part of *pool.xml* as an example.

```
<pool>
       <site>
              <name>AcmeAirlines</name>
              <datalist>
                     <data>Destination: Montevideo
                        Item Subtotal: $59.00
                        Total Before Tax: $59.00
                        Tax: $11.95
                        Total : $70.95
                   </data>
                     <data>Items
                        Traveller: 1
                        Ship: Catalonia
                        date:08/01/2000
                        Shipping time:7:40
                        Availablility: Yes
                        </data>
                     <data>Telephone: 01147762973</data>
              </datalist>
              <date>01/02/2001</date>
              <URL>https://www.acmebus.com/Orders</URL>
       </site>
...
```

Figure 4.2 Part of pool.xml, the XML document where data gathered remains

The client leaves collected data in an XML document. The client connects to the servers one a time and adds the corresponding information to the XML file. The user can then edit this file with a text editor as well as with any of the XML editor available. Furthermore, the data obtained can directly feed specific user's application.

Consequences

- The pattern enforces user and application integration and collaboration through the web.

- We get a modular solution that allows clean sharing of data between non-related components, avoiding one's interference in the other's affairs. In addition we don't interfere with internal data representation in the source, nor in the client.
- Increased flexibility in connections with information providers. As a client application can get data from foreign systems without interference, it can increase the number of information servers it can connect to. It does not need to change its code to connect to a specific server. It even could connect to non-standard devices without a significant change in the code.
- Wider target of data consumers for a site's data. Information provider sites wish to widen the target readership for their data. Again it could be possible to connect to other devices such as handheld, as long as this device has a matching component to connect to.
- No interference with the navigation design layer [Rossi+98]. We just add a token of the connection component.
- A flexible design for data format transmission and storing. We reutilize data keeping the format.
- Storing gathered data in an XML document allows us to add more data to the XML pool in another session, which lets the user have his data centralized as well as at hand.

Further comments

This pattern is a mediator in the sense of the GoF pattern Mediator [Gamma+95]. It resembles the mediator in that it serves as an intermediary that helps in interactions of a group of applications.

In the implementation we suggest the use of XML to share information because XML is an effective way to transmit structured information between non-related applications. If needed, the client and server could negotiate an XML data format. The server would give the client a specific data format. To improve implementation it would be possible to let the client connect simultaneously to several servers.

Also, we use XML as a standard to facilitate computer integration and collaboration. In the implementation, servers and clients are responsible for giving context information (if needed). That means that both kinds of objects have sharable information as well as context information. This resembles Gang of Four's Flyweight pattern [Gamma+95] in that the objects have intrinsic and extrinsic state. Intrinsic state is stored in the objects and it is information that is independent of the object context, and thereby sharable. Extrinsic state, on the other side varies with the object's context.

Furthermore, the use of XML helps language and platform independence. It brings flexibility at the moment of choosing the language, because as long as the servers offer data following the XML standard, the pattern's users don't care about the language they use. Besides, as shown in the example, we can develop an XML structure to store gathered data and read it from the XML document using a text editor, an XML editor or a custom application specially designed to process that data.

5. References

[Cagle00] Cagle, K., *XML Developer´s Handbook*, SYBEX, 2000.

[Gamma+95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns*, Addison-Wesley, 1995.

[Kay00] Kay, M., *XSLT Programmer´s Reference*, Wrox, 2000.

[Nielsen99] Nielsen, J., *Designing Web Usability: The Practice of Simplicity*, New Riders Publishing, 1999.

[Lifia xml]. http://www-lifia.info.unlp.edu.ar/xml/, LIFIA, Facultad de Informática, UNLP, Argentina.

[Rossi+98] Rossi, G., Schwabe, D., *An Object Oriented Approach to Web-Based Application Design*, Theory and Practice on Object Systems. Wiley and Sons, October 1998.

[Rossi+99] Rossi, G., Lyardet, F., Schwabe, D., *"Patterns for designing navigable Information Spaces" Pattern Languages of Programs IV*, Addison-Wesley, 1999.

[Rossi+00] Rossi, G., Lyardet, F., Schwabe, D., *Patterns for e-commerce applications*, EuroPLoP'2000, 2000.

[Saxon] Kay, M., *About Saxon*, http://users.iclway.co.uk/mhkay/saxon/instant.html, 1999.

[Schwabe99] Schwabe, ., D Rossi, G., Lyardet, F., *Improving Web information systems with navigational patterns*, Computer Networks 31 pp.1667- 1678, 1999.

[Spencer99] Spencer, P., *Professional XML Design and Implementation*, Wrox, 1999.

[W3CDOM] *Document Object Model (DOM) Level 2 Specification Version 1.0 Candidate Recommendation of the World Wide Web Consortium*, http://www.w3.org/TR/REC-DOM-Level-1, 01/10/98.

[W3CNamespaces] *Extensible Markup Language (XML) Specification Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org/TR/REC-xml-names, 14/01/1999.

[W3CXML] *Extensible Markup Language (XML) Specification Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org/TR/REC-xml, 10/02/1998.

[W3CXPATH99] *XML Path Language (XPath) Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org, 16/11/1999.

[W3CXSL] *Extensible Stylesheet Language (XSL) Version 1.0 W3C Candidate Recommendation*, http://www.w3.org/TR/xsl/, 21/11/2000.

[W3CXSLT] *XSL Transformations (XSLT) Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org, 16/11/1999.