

RivCom Limited
Lotmead Business Village
Wanborough, Swindon
Wiltshire, SN4 0UY. UK
Tel +44 (0) 1793 792000
Fax +44 (0) 1793 792001
www.rivcom.com

XML on the desktop: CGAT application

Introduction

The XML family of standards is changing the ways in which people interact with documents. Where traditionally documents have served as static snapshots of information, the combination of XML, XSL and Xlink will allow us to deliver documents that look and behave very much like interactive software programs.

A document designer now has to consider not only how the document should look, but also how it should behave, and how it will fit into the information architecture of which it is a part. Similarly, an information architect who is designing a system must consider which goals of the project are best served by traditional software development, and which should be filled by XML-based interactive documents.

At the WWW6 Conference in Santa Clara in April 1997, RivCom gave the first public demonstration of XML content being presented through the use of stylesheets in an industry-standard browser. RivComet, the technology used in that demo, has now been enhanced and is available as an ActiveX control running under Internet Explorer 4.0.

RivComet has been used to deliver a number of XML applications for Shell International and other companies. One of these, the *Competence Gap Analysis Tool*, is the basis of this presentation.

This talk demonstrates the application in action, then looks more closely at the XML structures on which it is based. Finally, it examines the issues that this kind of project raises, starting with the concrete "why did they do it this way?" and moving outwards to a generalized discussion of the strengths and weaknesses (and costs and benefits) of using XML-based interactive documents in an information architecture.

Application overview

The *Competence Gap Analysis Tool* was developed by RivCom for Shell Services International. The requirement was to create an interactive tool to allow people working in diverse units throughout Shell to

- rate the job competences required for different jobs within the organization, and save these ratings to disk for later reuse
- rate the competences of a particular individual performing that job, and optionally save them to disk
- view a 'gap analysis' comparing the individual's competences with the requirements of the job, in order to assess how well-suited they are to the job and to identify areas where training or improvement may be required.

The tool had to be data-driven, so that the supplied list of competences could be easily modified to handle jobs with vastly different requirements. And it had to be lightweight and capable of running virtually anywhere that Shell operates — which is all over the world.



These requirements made RivComet and XML an ideal solution. RivComet uses XML to store data, and stylesheets to govern the presentation of the data and the interaction available to the user. A single file downloaded from the server is presented in multiple ways and combined with data stored locally or entered interactively by the user. The result is a fully-fledged application that runs on the client, with network traffic kept to a minimum.

Functionality

The application's opening page presents a simple workflow consisting of three tasks:

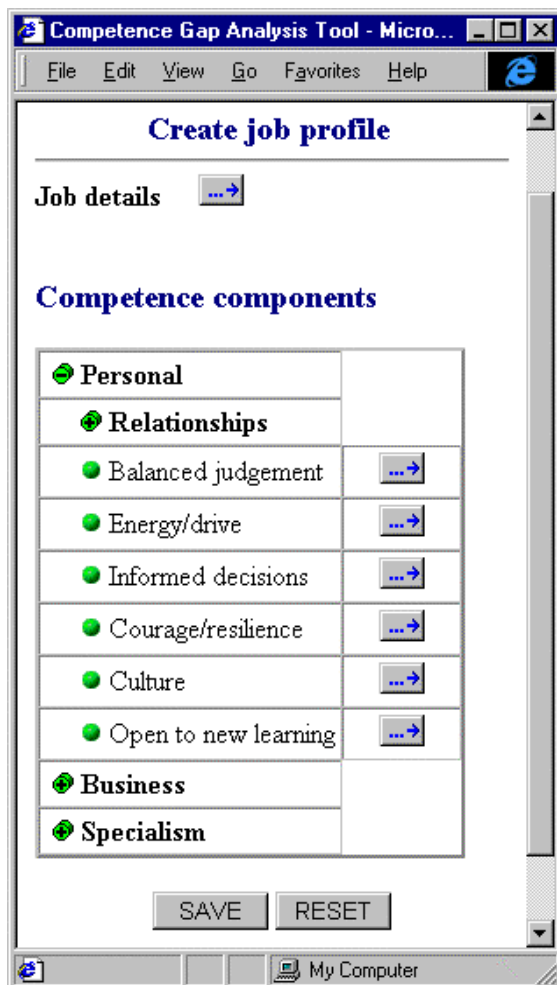
- Create/Edit job profile
- Create/Edit individual profile
- Gap Analysis.

Each task comprises a separate page, within which the competence framework can be browsed and interacted with in various ways. Working through these pages, the user performs the following actions:

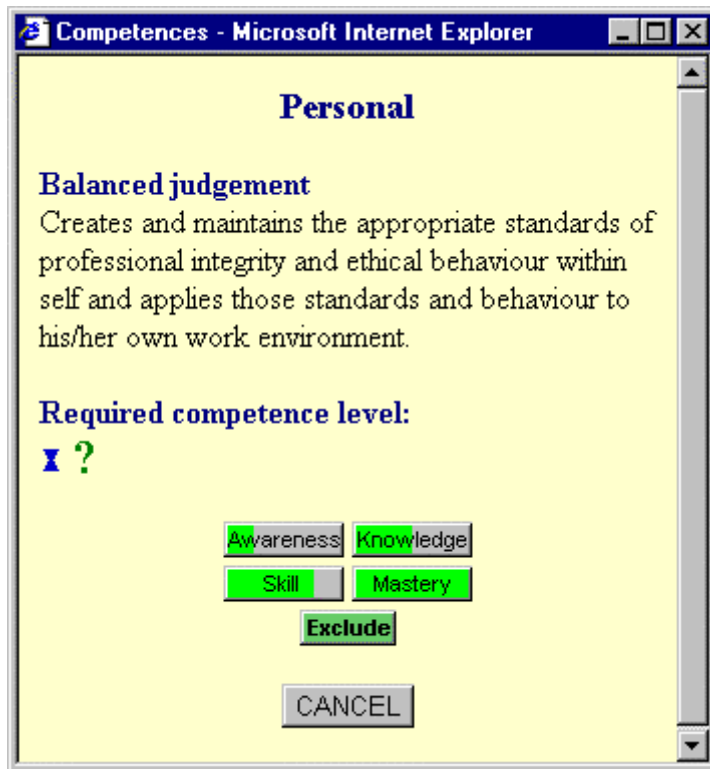
- Rate the job competences required for a specific task, and save these ratings on disk for later reuse
- Rate the competences of a particular individual performing that task, and optionally save them on disk
- View a visual Gap Analysis comparing in individual's competences with their job's requirements.

Step by Step

The job profile information is displayed as a list which the user can expand and collapse like an outliner:



The outline view shows only the names of the competences and competence groups. However, by clicking on the button to the right of each competence, a full description can be viewed in a popup window:

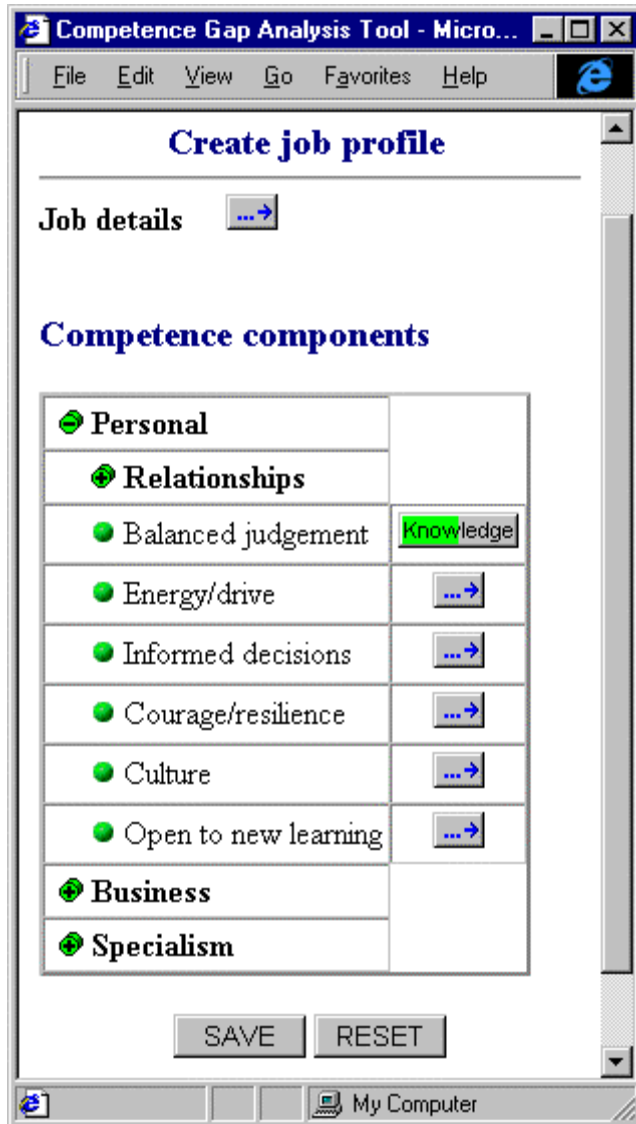


Below this description is a set of buttons corresponding to various competence levels, from **Awareness** to **Mastery**. These buttons may be used to specify the level of this particular competence that is required for the job whose profile is being created. Clicking one of these buttons causes an identical button to appear in the main window against that particular competence. The **Exclude** button indicates that a particular competence is not relevant to the job in question.

An additional feature of the popup window is the help text that is revealed by clicking on the blue double triangle to the left of the green question-mark icon.



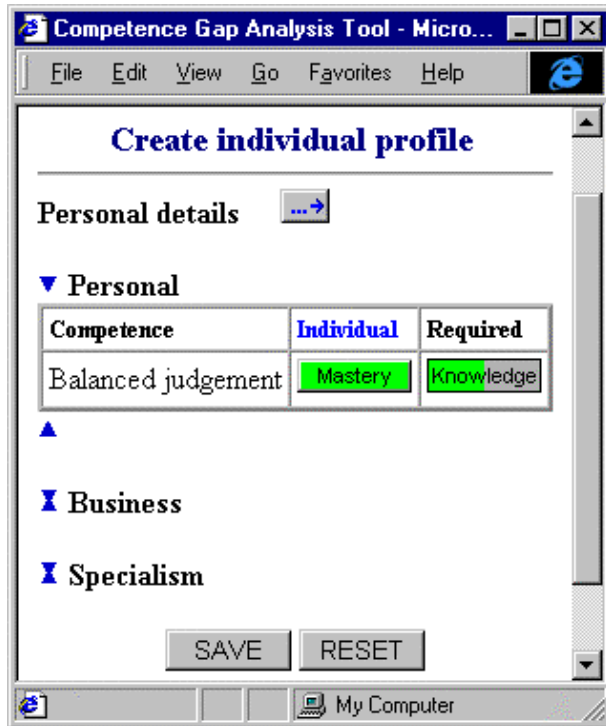
Once the user selects a competence level, their choice is transferred back to the main window:



If the user presses [Save] and stores their information to disk, the application creates an XML file that contains the requirements of the job they have described. (See below for an example.) The reason this file can be saved to disk is that the task of defining competences for a particular job is performed just once. Once the job definition has been saved, at any time and as often as necessary the user can create separate files for each person who is performing that job.

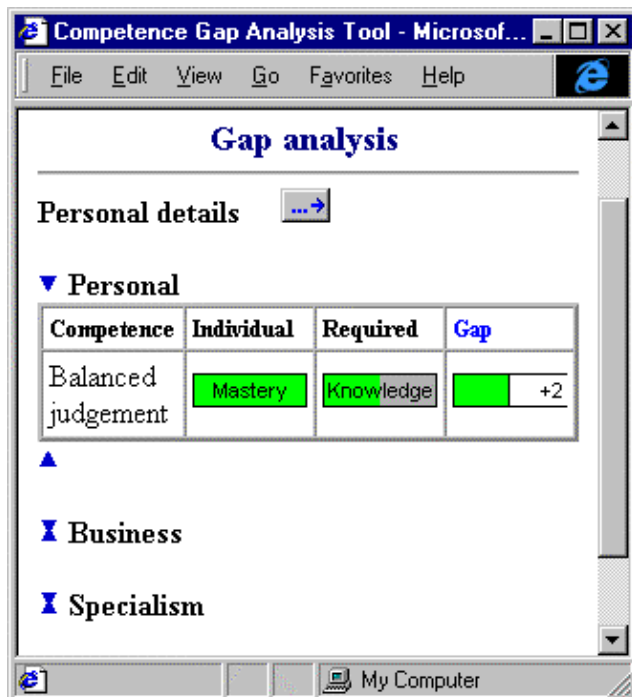


The task of defining a person's competences is done in another window which looks a lot like the one shown above, but has a column for the individual's ratings. Assuming the current worker has a "Mastery" of Balanced Judgement (and has not entered any other competence values), the display ends up looking like this:



The user can optionally save their personal competences for later review.

Finally, the tool contains a third window which displays a "Gap Analysis". This is a graphical comparison of the requirements for this job and the values entered by the current individual. The analysis can be performed either immediately after entering this individual's values, in which case there is no need to create the third document, or at any later time. The gap analysis for the above entry would look like this:



Had the gap been negative it would have shown in red. A full display featuring both red and green elements is quite colourful.

Behind the scenes

All this functionality results from different presentational styles being applied to structured XML information. The different competences and competence groups are nested XML elements that are displayed or hidden according to the style that is applied to them as the user clicks on the plus or minus buttons. The competence levels are displayed by a style that is responsive to the current value of the relevant attribute of the XML element, which is dynamically altered in memory in response to the user clicking on the relevant button in the popup window.

The stylesheets and formatting rules are written in a proprietary syntax developed by RivCom, as there is currently no W3C Recommendation for the application of presentation and behaviour to XML content. (And at the time CGAT was prepared, there wasn't even a Working Draft of XSL.) However, RivCom is represented on the XSL Working Group and will adopt XSL syntax as soon as it has been approved as a W3C Recommendation.

Use of XML

The application is built around a read-only file containing a set of nested XML elements (the competence framework itself) and a number of stylesheets and associated formatting rules that define the application's functionality.

The application allows the user to browse through the competence framework and assign 'required' and 'individual' levels to each competence.

Here is an extract from the XML representation of the competence framework:

```
<comp ID="cpcf1" required="0" indiv="0">
  <title>Delegation</title>
  <desc> Creates appropriate scope of work, authority, and schedules
  for staff and is able to delegate work in the confidence that at
  least the desired output will be achieved.
</desc>
</comp>

<comp ID="cpcf2" required="0" indiv="0">
  <title>Respect</title>
  <desc>Demonstrates a fundamental respect for, and a genuine interest
  in, people at work, understands individuals and provides caring
  support for them at times of need.
</desc>
</comp>
```

The zero values of the **required** and **indiv** attributes on the competences in the above sample indicate that no competence levels have yet been set for them.

When the user saves a competence profile to disk, a local file will be created that gives non-zero values to these attributes. For example:

```
<data>
  <comp ID="cpcf1" required="1" indiv="3"/>
  <comp ID="cpcf2" required="3" indiv="2"/>
</data>
```

In the demo version of the application, the 'save' function has been disabled. However, the modified values of the attributes are held in memory and applied dynamically just as if they had been included in the master file, or combined with it from a locally saved file.



As the above code fragment illustrates, one of XML's great strengths – the ability to separate content from format – has been fully made use of. The fragment stores competence levels in a very concise and readable way. It is for the stylesheets to determine the one or more ways in which this competence information is presented to the user.

Runtime Technical Architecture

The text files used in this application contain virtually no HTML. All of the HTML required for rendering the pages in the browser is generated on the fly by the RivComet™ ActiveX control. Here is an outline of what happens:

- The initial HTML page contains RivComet, a small piece of JavaScript, and some HTML consisting of a single empty DIV element.
- IE builds a DOM of the HTML, which is virtually empty. However, this is an important step because RivComet will make extensive use of the DOM.
- The JavaScript tells RivComet the name of the file that should be retrieved and displayed.
- RivComet uses Windows services (the Winlnet API) to retrieve the specified file.
- The file contains some XML content, a set of stylesheets that associate formatting rules with the XML elements in that content, and the formatting rules themselves, using RivCom's own declarative style language.
- RivComet parses the XML document and attached stylesheets and formatting rules, applying the rules to the XML in order to generate HTML that it writes dynamically into what was the empty DIV element in the DOM.
- IE then re-renders the page with the now fully populated DOM, and a full HTML page appears on the screen.
- The resulting page may contain buttons or hotspots which, when clicked, cause RivComet to launch a second window. Here again, RivComet generates new HTML content, dynamically, and inserts it into the DOM of the new window. As in the main window, this HTML content is determined entirely by the application of style rules to the structured XML content in the initial file or a combination of that file and other files.

The above architecture is extremely flexible and powerful. Of course, it uses a non-standard style language and an ActiveX control to achieve the desired results. But I believe (or at least hope) that before long XSL will support this level of interactivity, and the major industry-standard browsers will support XSL style sheets. At that point we will be able to deliver the functionality embodied in the CGAT via standardised a syntax and non-proprietary tools.

Issues and Conclusions

Why Did We Use This Approach?

This project came at the heels of several "more traditional" publishing projects in which RivCom had used XML and style sheets to publish the on-line versions of several sets of Shell's corporate documents. Therefore, at the time that Shell raised the issue of creating a Competence Gap Analysis Tool, the RivComet technology was hot off the presses and fresh in everyone's mind. Had it not been for this "mind share" factor, it is possible (even likely) that the client would have looked to traditional software development to address their requirements. Or, they might not have proceeded with the project.

Shell's requirements were that the tool should:

- be data-driven
- feature an easy mechanism to modify the underlying structures
- have minimal software installation and operational requirements (be "lightweight")
- have low implementation and operation costs.



After a short analysis, Shell decided to try a document-based approach because:

- The underlying technology had already been used in the other projects, so the incremental software development costs would be quite low
- The application would run in standard browsers using a plug-in that was already in use elsewhere in Shell
- Future improvements to this 'document-as-application' could be made with no software development at all, simply by modifying the stylesheets
- Everyone involved in the project was comfortable with the document paradigm, where the different components of the application (the source document and the additional information created while running it) would be stored as ASCII documents. (The issue of whether or not the additional information would be stored in XML format was not considered important.)

Would traditional software development have succeeded?

In this case we do not know. While the technology required to implement the CGAT in a more traditional form is trivially available, the costs of software development and the cultural attitude of the client might have led them not even to consider it.

In general, people assume that software development is complex and therefore costly, and that document publishing is simple and therefore less expensive. Of course, in practice both of these assumptions may be false: the cost and complexity of a given project is related much more closely to the requirements of the project and the nature of the information than to the technology that is used. (Unless, of course, one selects technology that is unsuited to the task.)

In this case, I believe that the division of Shell responsible for the CGAT was willing to undertake the project in part because of its low cost and low risk. Had we proposed to develop the application in VB or Java, the real costs would probably have been slightly higher, but more importantly, the client's fear factor would have been considerably higher. For another client, especially one more accustomed to commissioning software development, this would not have been an issue.

Decision Factors

There are a number of key factors that come into play when deciding whether or not to use a document-based approach in a given application.

Costs and Benefits

One must start with a traditional analysis of project costs compared to projected benefits. The document paradigm may be more or less expensive to develop than Java or VB applications, and this difference must be quantified and factored into the decision.

Client Attitude

As discussed above, the client may be inherently more comfortable with one of the paradigms. Given the complexities of any development project, this comfort factor can be as important as the hard data points.

Available Infrastructure

How will the application be distributed to its users? Can their desktops support the applications we develop? This is less of a factor within an enterprise that has standardized on a software desktop and has put mechanisms in place to distribute Java apps, ActiveX controls, plug-ins and the like. It is more of an issue when distributing to end users across the web, where about the only thing we may be able to rely on is that they have a web browser. (But which version of web browser, and what support it contains for the required technology, is of course an important issue.)



User Attitudes and Expectations

The support within browsers for interfaces that are similar or identical to more traditional software applications is rapidly improving. But to the extent that a document-based architecture results in a different interface than what users are accustomed to, one must evaluate not only the interface's absolute fitness-for-purpose, but also the user community's likely response to it. (Like the earlier point about client expectations, this is a subjective factor that can be just as important to a project's success as other, more quantifiable considerations.)

Source Formats

Even taking into account the benefits of XML as an information interchange and transformation medium, there are overheads involved in converting information from one form to another. The more closely the source information matches the XML format, the easier it will be to use a document-based architecture.

Security

The security model available in traditional software applications is quite different from (and generally more robust than) the browser-based security model. It is important to consider what level of security is required for the application, and whether it can be reliably implemented within the browser environment.

Links to Other Information

On the other hand, URL-based hyperlinking is an extremely useful and easy-to-implement mechanism for linking the information in an application to other corporate information. Here the fact that we are running in a browser with access to the Net gives document-based applications a real edge over other forms of application delivery. Links can be implemented in the initial delivery of the application, or added later by revising the style sheet and adding another layer of XML documents that map the content in the application to relevant external resources.

Conclusion

The world of publishing is changing rapidly. Soon, it will be common to deal with interactive documents that behave just like the applications that were written in Visual Basic or C++ just a few years ago. Whether or not this is a good idea for a given project is not clear-cut. The decision depends in part on the nature of the information being published, in part on the facilities and constraints of the project's infrastructure, and in part on the attitudes of the clients and users.

As the XML-related standards gel, as the built-in support for XML-based interactivity in browsers increases, and as users become more accustomed to this software paradigm, the boundary between documents and software applications will blur to the point of disappearing. Shell's *Competence Gap Analysis Tool* may be the first XML-based interactive document-as-application that we know of, but it surely won't be the last.

