CSRP 477: Architectural Requirements of a System Retrieving Adaptive Image Objects

Malcolm McIlhagga, Ian Wakeman
School of Cognitive and Computing Sciences
University of Sussex
Brighton BN1
9QH

September 29, 1998

Abstract

We provide a number of design choices which should be addressed when designing an adaptive application. These choices are common to all adaptive applications, and are fundamental in determining how the application is adapting to changing resources, while providing user utility. We have illustrated their use in the design of an Image Proxy for the WWW (Wakeman *et al*, 1997).

1 Introduction

There are a number of adaptive applications, ranging from those attempting to reduce congestion and limited bandwidth, such as vat (Van Jacobson and McCanne, 1992), vic (McCanne and Van Jacobson, 1995), ivs (Bolot, Wakeman and Turletti, 1994) and rat (Perkins, Hardman, Kouvelas and Sasse, 1997), to User Interface Management Systems such as Amulet (Myers, McDaniel, et al., 1997) which attempt to adapt to the display capabilities through battery management schemes within the operating system. The rise of middleware has generated renewed interest in providing generic adaptation policies, but we believe that the interfaces to manipulate resource requirements are phrased in the wrong vocabulary (Wakeman, McIlhagga and Ormsby, 1998). Recently there has been an attempt at defining object design patterns (Posnak, Lavender, Vin, 1997), but these do not address the application holisticly. The work on the Glomop architecture (Fox, et al., 1996) is closest in approach to our work. We have focused on developing a policy control architecture rather than developing a scaleable solution, and in particular in generalizing to an adaptive applications architecture.

2 Adapting to Constraints: Choices in Application Design

An adaptive application is one in which the application changes its behaviour according to the perceived constraints in the environment, so as to maintain the semantics of the application for the user.

If we take the above definition as our starting point, we can break the problem of designing an adaptive application into stages. The constraints in the environment should first be determined.

E.g. for mobile computing applications, the likely constraints are network quality of service, battery power and display capabilities.

We must next define the semantics of the application. What is the application trying to achieve? Having made a best guess at the semantics, our design goal is to ensure that the semantics of the application remain invariant across the constraint-reacting behaviours of the application, thus the utility of the application remains high. This leads us to a solution in which the implementation of the application should be opened up so as to allow the choice of some equivalent but less resource-hungry implementation. We thus come to our first design point -

3 Providing an Open Implementation of Components

The software engineering concept of open implementation (IO) (Kiczales, 1996) enables the designers of components to open up the implementation of their component so that it can be adapted to suit various needs. The behaviour of the component should be described by its interface abstractions, however the implementation is generally hidden. Following the design guidelines of Maeda (1996), the component must offer other meta-interfaces through which the programmer can adjust the implementation. If we are aiming to produce distributed applications which scale across networks and display capabilities, then OI offers an approach to enable scalability. Simply put, the OI part of a software component exposes the network requirements of a particular component's implementation. By allowing the adaptation controller to manipulate the implementation of a component, the controller can adjust the network requirements of the component and thus adapt the component to prevailing network conditions. Kiczale, et al. (1997) suggests that the designer of an open implementation interface should attempt to:

1. Define an abstract 'black box' interface, which instantiates the useful behaviour of the component.
2. Using the domain knowledge of both the inherent implementation of the component and of how clients will use the component, define interfaces which allow the client to control implementation strategies. The inherent implementation of the component is based around the abstractions that will be used to interface the component. In the case of the file caching mechanism described in (Maeda 1996), it is the disk buffers and caches of a file system. For our network retrieval architecture discussed later, the abstractions are the compression schemes and representations of the various forms of multimedia. Since the abstractions used in understanding the implementation of a component are closer to the domain of the application, the designer will be able to better tune the adaptation of the application.

3.1 Degrees of Freedom in Degradation Trajectories

The OI interfaces of the components will normally provide a discrete set of implementations upon some variation. Imagine a simple speech tool which provides a set of encodings to use for speech. As the degree of compression in the encodings gets higher, the bandwidth requirements decrease but the quality of the resultant speech is decreased, thereby decreasing the utility of the application. For more complex applications, each component may provide other choices of implementation which are orthogonal to each other; they reduce resource usage differently. If we were to add redundancy to the speech encodings, using some of our bandwidth to provide data to repair lost packets at the expense of greater latency, we would provide choices about how to use valuable bandwidth.

If we regard each component as supplying an OI interface which can be adjusted independently of the other components, the designer has a choice over each component of an implementation. The number of independent axes which contain implementation choices is the degree of freedom of the application. Adjusting the OI of a component can be viewed as placing the it at some new point in the implementation choice, resource usage, application utility space. As we select a new implementation choices to decrease a component's resource usage, we generally decrease the utility of the overall application.

Thus for each of the possible component implementation, we could in theory measure the utility of the resultant application. The designer must understand this space of possible implementations, for it is from it that they choose a degradation path matching resource variations.

3.2 How Transparent Should the Degradation be?

Having plotted the implementation space, the designer must next decide how the resource usage is to be monitored and controlled. For networked applications, this has generally been a congestion signal such as packet loss. Processor sensitive applications monitor, by examining the process queue size (Kouvelas and Hardman, 1997). The designer must next assess the availability of resources upon which to switch implementations, and whether to include the user within the loop. Systems which exclude the user and use closed loop feedback need to worry about the stability of the control loop, and the effect that varying utility will have on the user. Systems which include the user must ensure that the user is educated about the need for their collusion in adapting the application. Systems in the latter category include battery monitors asking the user about closing applications, or video conferencing system whose adaptation would reduce the quality to less than the minimum demanded (Bolot et al., 1994).

3.3 Run-time versus Design-time Behaviour Specifications.

Having investigated the implementation space and decided upon how degradation should be controlled, the designer must now determine when the possible trajectories through the implementation space are decided - when designing the application or through interpreting some set of instructions later on.

Design-time behaviours include the use of reactive protocols for congestion, such as those in the video conferencing tools vic and ivs. These have fixed trajectories through the implementation space. In general design-time policies are easier to encode.

However, as the applications become more generic and are used for more disparate tasks, the semantics of the application are increasingly determined by the context of their use, and the utility of the application is difficult to pinpoint. An application which simply reports the state of some object has simple semantics and will only be used in a very limited set of contexts can have it's behaviour pre-determined. Conversely, a web browser is used in many different ways (Light, 1998) and the semantics of the retrieved pages depend highly upon context. As designers, we can only make best guesses about the utility of various implementations, and so we can only select trajectories through implementation space that approximate the profile of some imaginary user. In real situations, we must allow users to override a generic profile to determine which aspects of the application semantics are important to them at run-time.

3.4 Selecting Fixed Trajectory Behaviours.

Users of applications don't care about the subtleties of network and display performance - they just want the software to work. Thus whether the behaviours are fixed at design-time or run-time, the user should be able to use the application without fiddling with editors to set-up values. The designer must determine and install trajectories through the implementation space that correspond to some expected path through resource availability. In doing so determine likely contexts in which the application is to be used, approximate the semantics, and then experiment with selecting behaviours.

The choice of fixed trajectories has a large impact on whether users will use an application and so is a very important part of the design process. We believe that the context and semantics can only be determined by users, so as designers we must bring users into the design process. If possible, we should study them in situ., but if it is not feasible they should be studied in experimental mock-ups.

3.5 Designing Appropriate Interfaces for Customization.

Traditionally, customization of network and distributed applications has been through dialogues which provide direct access to program variables. However, this is of limited use when the abstractions are themselves complex and have little or no meaning to the user.

The OI of the components provides abstractions in the application domain to allow manipulation of behaviour. However, these abstractions may be far removed from the experience of the user. In particular, the nuances and abstractions of network and distributed applications are rarely understood by programmers; it is unlikely users will grasp specifications in terms of network abstractions.

Instead, the abstractions of the application must be mapped from the system image onto metaphorical controls which build upon the experiences of the user. By using such metaphors we can educate the user to have an appropriate and useful mental model of the application (Clark and Sasse, 1997).

4 Run-time Policies: The Media Policy Description Language.

Win our image proxy server we have chosen to implement the policy of degradation at run-time for the following reasons:

- The underlying technology has four main methods through which the implementation of the transferred media can be manipulated. The application thus has at least four degrees of freedom in which we can change the implementation, and so providing a wide choice of trajectories of degradation.
- The application using the media is very generic - the Web model of documents is being pushed by Microsoft and others as the basic metaphor for the next generation of machine interfaces. Since the uses of the application are legion, we as designers cannot constrain the choice of policy degradation.
- The ability to transform the media entirely across types forms an even wider implementation space. Since transformations may be chosen as more convenient by the user, e.g. a partially

sighted user may want text to audio, we believe that users determining adaptation policies provide an exciting use of the technology.

- There are two associated actors interested in how a media instance is downloaded and displayed - the reader and the author of the media. Each of these actors may have preferences on how media should arrive, users may want it as fast as possible, yet authors may require colour. It is easier to resolve conflicts at run-time.

Our policy degradation objects are written in the Media Policy Language mpl and are interpreted within the application and the server. They can be merged to form new policies representing the combined needs and experience of user, author and system designer.

The policies have been used to control the preferences users have as to how images are viewed across the Web. However, the principle applies equally to other multimedia application (networked or otherwise) and indeed to any application that wants to scale it's interface in some way. User, authorial and default policies could be devised to allow an application's widgets to present themselves in a sensible manner on differing platforms and visual displays.

mpl is a rule based language that allows the mapping of certain actions to specified groups of media according to the current networking and display conditions.
The general format is:

```
path : condition : action [, submission]
```

It means "apply action to the specified media type (path) if the condition is true and some rule belonging to a different rule group doesn't override the path". Conditions are legal Boolean expressions.

We use environment variables to hold values of networking and display conditions and attributes of the media, as determined by the run-time environment (Sharples 1997). Each environment variable has a unique name and has an associated type. Currently variables are:

```
FILESIZE -       int
MEDIA-HEIGHT -   int
MEDIA-WIDTH -    int
MEDIA-DEPTH -    int
DISPLAY-WIDTH -  int
DISPLAY-HEIGHT - int
DISPLAY-DEPTH -  int
BANDWIDTH -      int (in Bytes/s)
RATE -           int (flow rate)
MIME -           string
META -           string (used for passing any other info.)
```

Actions are compress (lossless), reduce (quality: lossy compression), scale (dimension), and transform (from one media to another) etc.
For instance, if a user wishes to compress all objects over 10k:

```
media.* : SIZE >= 10*1024 : compress;
```

or if a user has a monochrome browser:

```
media.image.* : true : toMono;
```

When the browsers is always used over low bandwidth links:

```
media.image.jpeg : meta(progressive) == false :
toProgressive, submit default;
media.* : true : scale 75, submit author;
media.* : true : reduce 50, submit author;
media.* : true : compress 10;
```

Policies can be combined with environmental information to select paths of degradation depending upon the available constraints, so if the user wishes to ensure that all download times are less than five seconds, they first attempt to reduce the quality, then if this fails, they scale the object, using the resolution described in the next section.

```
media.* : SIZE/BANDWIDTH>5 : reduce 50;
media.* : SIZE/BANDWIDTH>5 : scale 75;
media.* : true : compress 10;
```

An author's policy to make a JPEG fit in the available space:

```
media.image.jpeg."www.site.org/pics/mypic.jpeg":
MEDIA-WIDTH > DISPLAY-WIDTH : scale (MEDIA-WIDTH
/ DISPLAY-WIDTH);
```

4.1 Policy Resolution

Policy scripts are compiled together to create a single executable policy. The policy can act on the multimedia object with which it is associated through the various interfaces discussed earlier.

When the policy is activated (asked to transform its media), it goes through a number of parsing and resolution phases. These determine which rules are relevant to the multimedia object, which rules can be removed or overridden by others and they establish a definitive precedence order between rules from different policy sources.

Initially any rule which applies to media other than that of the attached multimedia object are removed. Then rules which clash are resolved according to the following criteria:

1. user rules have precedence over authorial and default rules, except when 3. or 4. is in place,
2. authorial rules have precedence over default rules, except for 3.
3. user and authorial rules can specify submition to default rules.
4. user rules can specify that they submit to authorial rules. Four phases of multi-pass rule activation then take place:
    - Transformation involves firing rules that transform the object to another type in the media hierarchy. Rules that utilize the transformation interface are key to the writing of policies that cope with difficult display attributes and location dependant data.

- Reduction involves firing rules that utilise the reduce interface; thus reducing the quality of the attached multimedia object and so improving down load time. The reduction phase is a multi-pass operation. Each pass of the rules further reduces the multimedia object. Passes are repeated until no rule is fired, that is, all of the media's size and quality requirements are met. Of course this may never happen! So, the multi-pass mechanism is constrained by a set of heuristics which can identify looping, the exhaustive limits of compression and rules which reduce the multimedia object to the edge of our perception.
- Scaling is the process of altering the dimension of the Media. Images are scaled in the X,Y dimension, as is Video. For some media types scaling is not meaningful. It is not meaningful to scale ASCII and it is only meaningful to scale Audio in terms of amplitude or tone, qualities that can be adjusted to suit the user, but do not effect download time.
- Compression is a simple one-pass non-lossy compression. Media that benefit from this are those which, unlike JPEG and MPEG, do not support their own compression. E.g. any mime:text/*, can benefit from non-destructive compression. This multi-phase multi-pass process of resolution is at the heart of what makes the policy work. It combines the disparate needs of user, author and the system designers.

5 Fixed Trajectories: Determining a Default User Policy

There are three good reasons for the existence of a default policy. First, it is necessary to provide a reasonable behaviour for an object if no policy is provided by user or author. Second, it is sensible to insist on certain behaviours unless the user or author overrides them. It is sensible to use lossless compression on objects that implement no compression of their own, such as text, HTML, postscript, etc. We want to be able to provide default policy decisions that only effect the quality of the media slightly but massively improve download time or display usability, such as a small reduction in the quality of a JPEG or the frame rate of video. Finally, the default policy is the basis from which the user can incrementally develop their own policies.

6 Policy politics: Use and Usability

As system designers, we expected that developing an application that attempted to maximise utility under resource constraint would unconditionally be "a good thing". However, for shared applications in which the parties do not necessarily share the same goals, issues arise about which of the parties should retain power.

The degree of control that authors of Web pages have over presentation compares unfavourably with that in most other media, as the standard layout tool, HTML, is a logical mark-up language. HTML is changing slowly to take in more optical factors, such as specified fonts and styles, because of producers' dissatisfaction with this state of affairs. However, due to variation in browser interpretations these tags do not guarantee what is produced is what we see. And users continue to have the opportunity to override some of the layout tags, to switch off images and Java Applets.

As originally intended the balance of power between user, technology and author is a negotiable feature of the Web; and in continual flux. On the face of it, policies strengthen the user's hand and contribute to an uncertain environment for the author. If some authors continue to show the same cavalier attitude to image file size and the provision of "Alt" tags as a text alternative, then at least users have redress now.

But policies have been designed to allow either side to claim or relinquish control. An author must defer to user preferences where they exist and can only request certain standards of presentation. Conversely, a user may define a policy that is in part overridden if an author has produced a policy and in part defines their definite display and download needs. Thus, they can receive something close to the intentions of the author (if they so wish) or something that is "now readable" on their palm top. In the final analysis, if both parties have conflicting policies, then the user's must triumph. After all, it is the user that must wait while images download and the user who has such special needs as small displays or disabilities that necessitate differing policies. Our prototype software is a workable alternative to browsing with the images disabled and therefore goes a long way to ensuring that the author's vision is delivered, if only slightly modified.

An interesting philosophical perspective of the conflict between the author and user policies is that we are seeing a concrete representation of the post-modern clash over control of the text. Distribution of media over networks has provided another form of distance between the author and the reader, where the network and display may change the experience wished for by the author into something completely different, even before the reader brings themselves to bear. By making the changes forced by networked delivery explicit and configurable, we enable the author and the user to enter into a dialogue about what the media is intended for, and use policy precedence to resolve the asynchronous dispute harmoniously. But this dialogue can only occur if authors account for various alternate representation to their work.

7 To Conclude

In the development of an adaptive application following some sort of design criteria that incorporates the user is fundamental to the success of that application in optimising it utility in the face of an uncertain environment. The use of open implementation in our component design allowed integration of policy code into the application; the application now has a mutable trajectory through implementation space. This maintains the utility of the application to the user through the optimal use of available resources.

We are currently extending the Image proxy to a general proxy architecture in which any media can become active. We are also investigating authoring tools that encourage the generation of alternate representations of multimedia presentations, and that allow authors to express their policies.

8 Bibliography

L. Clark and M. A. Sasse (1997). Conceptual Design Reconsidered - the Case of the Internet Session Directory Tool. In: *People and Computers XII: Proceedings of HCI'97*, August 1997, pp. 67-85, Bristol.

Armando Fox and Steven D. Gribble and Eric A. Brewer and Elan Amir (1996). Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In: *Proc. Seventh Intl. Conf. on Arch. Support for Programming Languages and Operating Systems* (ASPLOS-VII)", October 1996, Cambridge Ma.

Ian Wakeman, Malcolm McIlhagga and Andy Ormsby (1998). Signalling in a Component Based World. In: *Proceedings of the First IEEE Open Architectures for Signalling.* April 1998, San Francisco, Ca. Van Jacobson and Steve McCanne (1992). Visual Audio Tool - vat Manual Pages.

G. Kiczales (1996). Beyond the Black Box: Open Implementation. In: *IEEE Software*, January 1996. Gregor Kiczales and John Lamping and Cristina Videira Lopes and Anurag.

Mendhekar and Gail Murphy (1997). Open Implementation Design Guidelines. In: *Proceedings of International Conference on Software Engineering*, May 1997, Boston Ma.

Isidor Kouvelas and Vicky Hardman (1997). Overcoming Workstation Scheduling Problems in a Real-Time Audio Tool. *Proceedings of Usenix Annual Technical Conference*, 1997, Anaheim Ca.

Ann Light (1998). *Interactivity on the Web.* http://www.cogs.susx.ac.uk/users/annl/tax.html.

Chris Maeda (1996). A Metaobject Protocol for Controlling File Buffer Caches. In: *Proceedings of ISOTAS* '96.

Steven McCanne and Van Jacobson (1995). vic: A flexible framework for packet video. In: *Proceedings of ACM Multimedia*, November 1995, San Francisco Ca.

Brad A. Myers and Richard G. McDaniel and Robert C. Miller and Alan S. Ferrency and Andrew Faulring and Bruce D. Kyle and Andrew Mickish and Alex Klimovitski and Patrick Doane (1997). The Amulet Environment: New Models for Effective User Interface Software Development. In: *IEEE Transactions on Software Engineering*, June 1997, 23 6, pp. 347-365.

Colin Perkins and Vicky Hardman and Isidor Kouvelas and Angela Sasse (1997).Multicast Audio: The Next Generation. In: *Proceedings of INET* 97, June 1997, Kuala Lumpur, Malaysia.

Edward J. Posnak and R. Greg Lavender and Harrick M. Vin (1997). An Adaptive Framework for Developing Multimedia Software Components. In: *CACM*, October 1997, 40 10, pp. 43-47.

Nick Sharples and Ian Wakeman (1998). *Netbase: Gaining access to Internet Quality of Service from an Application.* Technical report: CSRP 476. School of Cognitive and Computing Science, University of Sussex.

Jean Bolot, Ian Wakeman and Thierry Turletti (1994). Multicast Congestion Control in the distribution of Variable Bit Rate Video in the Internet. In: *Proceedings ACM SIGCOMM94*, August 1994.