UNIVERSITY OF SOUTHAMPTON

DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

ACHIEVING USER INTERFACE HETEROGENEITY IN A DISTRIBUTED ENVIRONMENT

Jonathan Dale

University of Southampton

Technical Report No. 96-1

April 1996

ISBN: 0-854-325-816

Achieving User Interface Heterogeneity in a Distributed Environment

Jonathan Dale

Multimedia Research Group

Department of Electronics and Computer Science

University of Southampton

UK

jd94r@ecs.soton.ac.uk

The introduction of distribution into the field of computing has enhanced the possibilities of information processing and interchange on scales which could not previously be achieved with stand-alone machines. However, the successful distribution of a process across a distributed system requires three problems to be considered; how the functionality of a process is distributed, how the data set on which the process works is distributed and how the interface that allows the process to communicate with the outside world is distributed.

The focus of the work in this paper lies in describing a model that attempts to provide a solution to the latter problem. The model that has been developed allows the functionality of a process to be separated from and to exist independently from its interface and employs user interface independent display languages to provide distributed and heterogeneous user interfaces to processes. This separation also facilitates access to a service from diverse platforms and can support user interface mobility and third-party application integration.

The goals and advantages of this model are partially realised in a prototype that has been designed around the WWW and its associated protocols, and it is predicted how the model could be fully realised by adopting a modular and object-oriented approach, as advocated by the Java programming environment.

1. Introduction

When a process is designed for a single processor machine, it is generally written with a specific platform and user interface environment in mind, for example, a Silicon Graphics ELF executable, a PC Windows executable, etc. Generally, these types of processes are not written with any form of distribution in mind, so the code to handle the user interface of a process and the code to handle the functionality of a process are written together, side-by-side.

Yet, even if good programming practices are employed to ensure that the user interface environment code remains separate from the functionality code of a process, this still means that the user interface part of a process must be rewritten for each user interface environment (Windows, X-Windows, Macintosh, etc.). The next logical step is to abstract user interfaces by adopting a portable API that is supported across a range of user interface environments. This would permit identical user interface code to be resused without modification and would provide a consistent representation of user interfaces across platforms. However, this does not provide a solution for distributed or mobile user interfaces.

Therefore, when considering heterogeneous *and* distributed user interfaces to processes, a number of points need to be taken into account:

- *User interface distribution*. When a process is executing within a distributed environment, it will generally need to make some form of interaction with the agency that launched it. If this agency is a user, then the distributed process must be able to redirect its user interface onto the display of the machine where the user is located. Similarly, the subsequent interaction from the user needs to be conveyed transparently back to the distributed process for processing.
- *User interface abstraction*. There needs to be a distinct separation between the code that handles the user interface and the code that handles the functionality of a process. This abstraction should be

sufficient to give a process the ability to choose dynamically a user interface that suits not only the intended environment (text-only, Windows, X Windows, etc.), but also the requirements of its programming (text-only representation, 2-dimensional representation, 3-dimensional representation, etc.).

- *User interface remote control*. If the agency that launched a distributed process is another process itself, then there is a need for processes to be able to communicate with each other. This communication can take two forms; the first is where a process masquerades as the user interface of another process, in order to control its processing; the second is where a process masquerades as the functionality of another process, in order to control its user interface. This would give a user (or another process) a meta-level control over their processes, for example, they could send a message instructing all user interfaces to close or to redisplay themselves on another display.

The X Windows protocol [11] goes some way to a solution to user interface distribution by allowing X clients to redirect their displays across the network to other X-servers in a transparent fashion. However, the X-Windows protocol is unsuitable to use as a general solution, due to the fact that only the protocol data is being redirected; the processes themselves can only execute on X supporting machines and the user interfaces can only be rendered on X supporting machines. Moreover, the X protocol does not integrate well with other user interface environments; a user on a Macintosh would expect user interfaces to be rendered in terms of the Macintosh user interface environment, not in terms of the X-Windows user interface environment.

The purpose of the research that has been undertaken at the Multimedia Research Laboratory within the University of Southampton was to design a model that addresses all of these problems. The distributed user interface system that has been developed allows the functionality of a process to be separated from its user interface, provides for the user interface of a process to execute within a variety of user interface environments and supports user interface redirection and process remote control.

Section of this paper describes how a process can be abstracted into a user interface segment and a functionality segment and details the protocols that must exist to allow them to communicate. Finally, section 2 shows how this model has been put into practice through the use of an experimental prototype.
Separating User Interaction from Data Processing

To give a distributed process the flexibility required to dynamically select new user interfaces to suit a specific user interface environment and to allow user interfaces to be directed across a distributed system, a process needs to be split into three separate but communicating physical processes; a *Process Handler* (PH) to manage the functionality of the process, an *Interface Handler* (IH) to manage the user interface of the process and an *Interface Viewer* (IV) to render user interfaces and process user input data (figure 1).

The motivation behind this dissolution is as follows. The PH is kept separate from the IH so that it does not have to be concerned with either the user interface environment or the IV that is selected; it is responsible for constructing user interfaces and processing user input data. The IH is kept separate from the PH and the IV so that it does not become involved in either the construction of user interfaces or the user interface environment; it is responsible for selecting and launching an appropriate IV on behalf of the PH and for returning user input data back to the PH. The IV is the only process which is user interface environment dependent, since it has to render the user interfaces supplied by the PH in a user interface environment-specific manner (for example, in X-Windows). The IV also accepts user input from the user.

This separation affords a distributed user interface process three features. Firstly, the process can

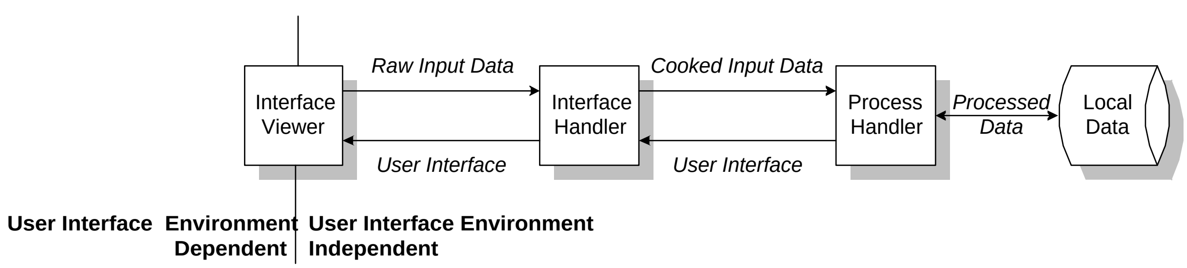

Figure 1 : Data Interactions between the Process Handler, Interface Handler and Interface Viewer

distribute its user interfaces across a distributed system, since the IH can launch an IV on the machine where the user is located. Secondly, the process can abstract its user interfaces from any particular user interface environment by communicating in a user interface independent display language, such as the Hypertext Mark-up Language (HTML) [3], the Virtual Reality Modelling Language (VRML) [2] and the Tool Command Language/Toolkit (TCL/TK) [10]. And finally, the process can allow its individual control processes (the IH and the PH) to be remote controlled by other control processes through the communication interface that exists between the IH and the PH.

The interaction between these individual processes can exist in two directions:

- *PH to IV communication*. The PH is responsible for generating new user interfaces, which it communicates to the IH. When the IH receives a user interface, it selects an appropriate IV and invokes it with the user interface to render.
- *IV to PH communication.* Upon rendering a new user interface, the IV accepts raw input data from the user and passes it back to the IH. The IH performs pre-processing (that is, *cooking*) on this data to convert it into a format that the PH can understand. Upon receiving this data, the PH processes and acts upon it, probably generating a new user interface for rendering.

This model assumes that the user interface has no state. Where user interface state must be retained, for example during mobility, it can be encapsulated in an intermediate 'gateway' process. An example of such a gateway is illustrated by the user interface system developed for the On-line Public Access Catalogs (OPAC) system [1].

To abstract both the PH and the IH from the destination user interface environment, the PH must be able to express user interfaces in a user interface independent display language. By adopting a core set of specific user interface metaphors, these languages provide a flexible and consistent set of display components, whilst not being constrained to a particular user interface environment.

The communication protocol that exists between the PH and the IH is based upon an event-driven messaging system, where each message consists of an action descriptor and associated data. Action descriptors map directly onto the functions that each process can perform. The employment of action descriptors also allows messages to be inserted into the message queues of either the PH or IH, thus allowing them to be controlled remotely by other processes. Figure 2 illustrates a sample message communication that could take place between a PH and a IH. Here, the PH begins by starting an IH and waits for it to become ready. After this has occurred, user interface independent display language negotiation takes place, with HTML being the accepted language. The PH then sends a user interface, specified in HTML, to the IH for displaying. The IH launches an appropriate IV and the user interface is rendered onto the user's display. Some form of user input takes place, which is pre-processed and returned back to the PH, via the IH. In response to some processing upon this data, the PH generates a new user interface, which is passed back to the IV. This cycle continues until a quit action is entered by the user, whereupon the IH kills the IV, confirms the quit action with the PH and terminates itself.

IV selection is based upon a negotiation procedure between the PH and the IH; the PH communicates a preferred user interface independent display language (HTML, in the previous example) to the IH and the IH returns an acceptance or rejection message, depending upon whether the display machine of the user can support it in terms of IV availability and graphical power. This negotiation continues until either a suitable representation is agreed upon or the default text-only representation is used.

In summary, the abstraction approach described in this section has a number of advantages for distributing user interfaces:

- Distributed user interface processes can execute on a wider range of machines if they are written in a portable language, such as C, since the PH and IH are user interface environment independent. Only the IV needs to be user interface environment aware.
- Distributed user interface processes have a finer control over how and where user interfaces are displayed, since they can spawn IVs on specific machines, yet they themselves can execute anywhere within the distributed system. This has the benefit that if a user moves interactively from one machine to another, it is simply a case of the user's IHs closing their respective IVs and restarting them on the user's new display machine.
- Distributed user interface processes have greater flexibility in which user interface independent

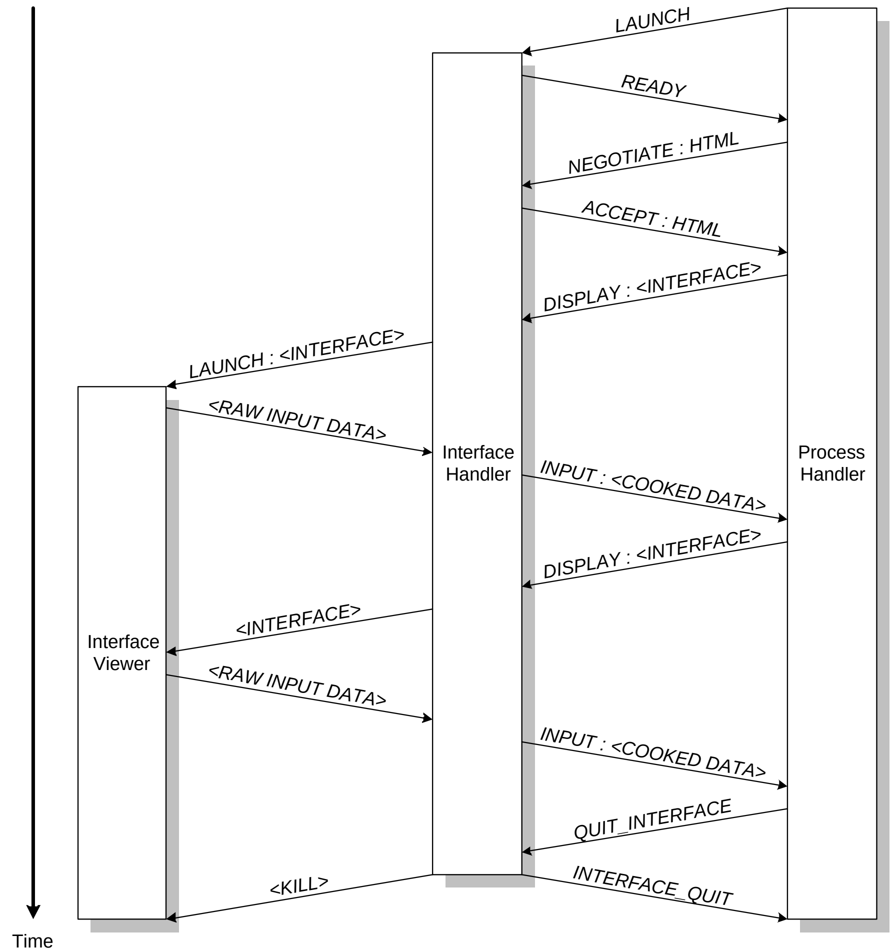

Figure 2 : Message Interactions between the Process Handler, Interface Handler and Interface Viewer

display language they decide to communicate. For example, a PH could generate user interfaces in three different user interface independent display languages (text-only, 2-dimensional graphical and 3-dimensional graphical). The power and graphical capabilities of the end-user's display machine (possibly specified by a user preference) and availability of an appropriate IV will determine which representation is actually used.

- Distributed user interface processes can be developed faster, since both the IH and the IV can be reused; only the functionality of the PH needs to be rewritten for a new process. However, if new user interface independent display languages are to be adopted, these must be built into the IH so that it can handle them.

- Distributed user interface processes can be controlled remotely from other processes by inserting messages into the queues of either the PH or the IH. They can also be replaced by proxies which act on behalf of the user interface to support or replace the user, for example, a 'bot' in Internet Relay Chat (IRC).

2. Achieving User Interface Management

The prototype that has been developed to allow experimentation of the ideas expressed by the model described in the last section is currently being used as part of the development of Microcosm: The Next Generation (MCMTNG) [8]. MCMTNG is an extension of the Microcosm open hypermedia system [5] into a distributed environment. The inclusion of the distributed user interface model provides user interface distribution across the Internet through the Hypertext Transport Protocol (HTTP) [4], user interface abstraction through the use of the Netscape Navigator [10] and HTML, and user interface remote control through the MCMTNG message routing system [7]. Figure 3 shows how the prototype

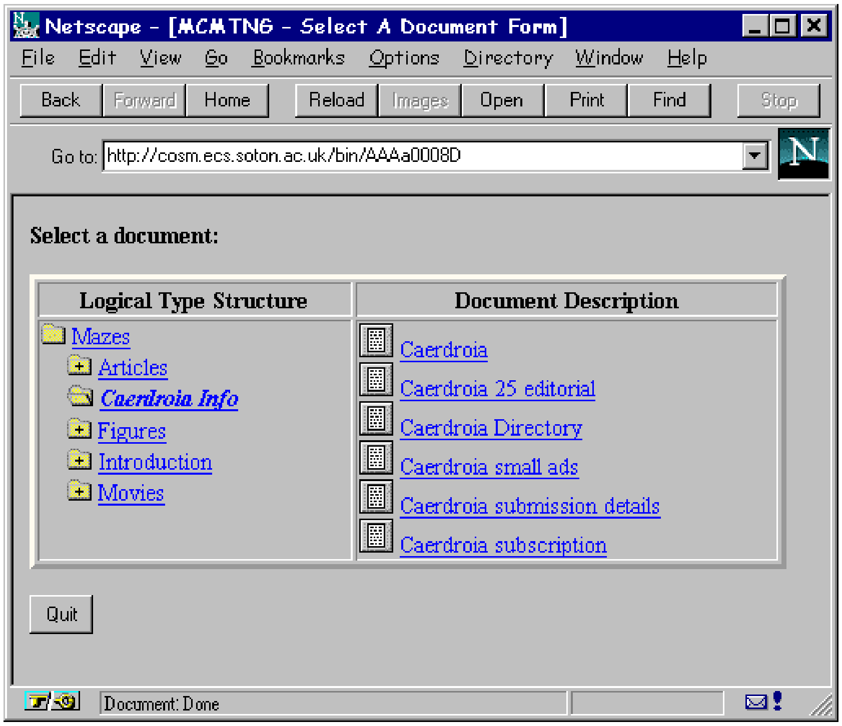

Figure 3 : The MCMTNG *Select A Document* **Process**

has been employed to build a simplistic file manager-like process (called *Select A Document*) which provides a 2-dimensional graphical interface onto a collection of hypermedia documents within the MCMTNG system. A user can click on a directory folder button in the left-hand pane to expand or collapse directory branches, select a directory entry to display the contents of that directory, or select a document entry in the right-hand pane to launch a *Viewer* process to view that document.

Every process in the MCMTNG system that needs to present a user interface is expressed in terms of an IV, an IH and a PH. Due to the fact that the IV and the IH can be reused for every PH, the development time for writing a new process is reduced, as a programmer only needs to write the HTML user interfaces and the corresponding PH functionality code segments to process these user interfaces. If each distributed process had to be written separately for each platform and for each user interface environment, then the development time of the MCMTNG system would have been much greater.

2.1. Distribution of User Interfaces

To distribute user interfaces across a distributed and heterogeneous environment, the transport protocol favoured by the World Wide Web (WWW), was used. HTTP is based around the mechanisms of Uniform Resource Locators (URL) which attempt to abstract the address of a document from its physical location, and WWW servers which attempt to resolve URLs to the physical location of a document.

The HTTP protocol is employed when a PH wishes to communicate a new interface to an IV (figure 4). The PH generates a user interface in HTML which it forwards to the IH (1). Upon receiving this, the IH writes the user interface to the file system of the local WWW server (2) and forwards its location (a URL) to the IV (3) for retrieval and subsequent rendering (4-6). Raw user input data is communicated back to the IH via a Common Gateway Interface (CGI) script (7), which organises the raw input data into a manageable format for the IH to receive (8). Finally, the IH pre-processes this input data into a form that the PH can use (9).

The use of WWW servers has the advantages that user interface documents can be retrieved transparently to the IV and can contain WWW hypertext links and other document types that are supported by the IV. The drawback, however, is that the prototype is reliant on a WWW server being local to the IH and accessible by the IV.

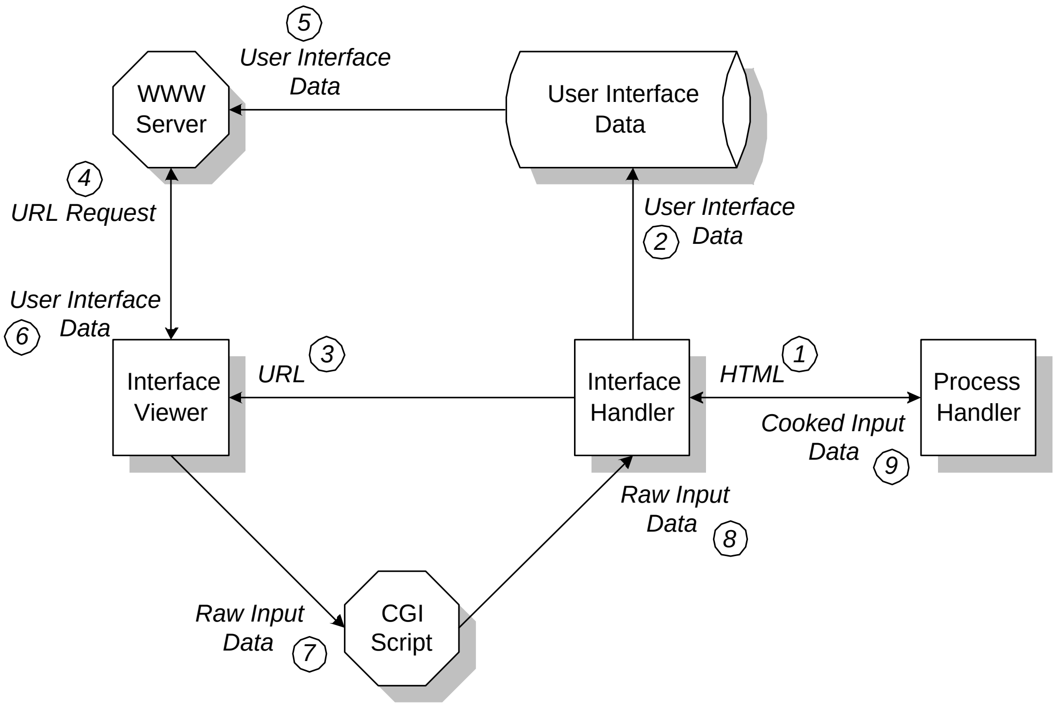

Figure 4 : Data interactions between the Distributed User Interface Prototype

2.2. Abstraction of User Interfaces

User interface environment information has been abstracted from both the IH and the PH through the adoption of the user interface independent display language HTML. HTML is widely used throughout the WWW and is an extensible user interface independent display language that can specify user interfaces in terms of components that are found commonly in user interface environments, that is, buttons, radio buttons, text gadgets, drop-lists, etc. In using an HTML-specified user interface, the PH can generate both static user interfaces that have been pre-written and do not need to contain active data, and dynamic user interfaces that are constructed on the fly from HTML components and active data.

The main IV of the MCMTNG system is the Netscape Navigator, which is used to render HTML-specified user interfaces. Netscape is a flexible and extensible IV that can be configured to handle multiple data formats (bitmaps, sound, etc.) and can also interact with WWW servers through the HTTP protocol to retrieve documents and data from across the Internet.

It is important to note that both other user interface independent display language and IVs can be supported by the prototype, through the negotiation stage that occurs between the IH and the PH, and by making the IH aware of a new IV. However, the IH currently assumes that an IV can be launched with a user interface on the command-line initially and will accept subsequent user interfaces and return user input data through an socket-based connection. If a new IV did not support this, then parts of the IH would have to be rewritten to accommodate the new communication method. The ambiguity of how data is communicated to and from the IV in a user interface environment independent manner is the largest problem with the prototype at the moment.

2.3. Remote Control of User Interfaces

The communication protocol between the IH and the PH was designed using an event-driven message passing system. This protocol, however, needed to be flexible enough to allow messages to be extensible and dynamic so that they could be used to allow any type of data to pass between the IH and the PH. The Microcosm open hypermedia system is based around a set of autonomous, communicating processes that use a message passing system to communicate. Messages are constructed from *tags* and associated *tag bodies*; a tag provides the index onto its free-form data tag body. The advantage of this system is that messages are dynamically extensible, because new data can be added by including a new tag and tag body. Additionally, a process does not need to understand the entire message to be able to process it; it can simply extract the tag bodies of the tags in which it is interested. Action descriptors are represented by a special tag, called \Action, which contains the type of event that has occurred, and the IH and the PH can determine the nature of the message by interrogating the contents of this tag. Similarly, data is passed within a message by using the \Data tag. Table 1 gives the complete list of the typical communications that take place.

As each action descriptor maps directly onto an event entry function in the IH or the PH, other processes (such as IHs and PHs) can influence their behaviour by sending them messages containing appropriate \Action tags. For example, in the MCMTNG system, there is a process which controls the invocation, configuration and destruction of processes, called the *Process Manager*. When a user wishes to end a session, the Process Manager sends a *Quit_Interface* action to every IH and PH, which has the effect of killing all of a user's processes. New remote control actions can be added by introducing new action descriptors and developing the corresponding code within the IH or the PHs.

IH to PH			**PH to IH**		
\Action	\Data	Purpose	\Action	\Data	Purpose
Data_Submitted	<user input data>	Submit user input data from the IV to the PH.	*Display_Interface*	<user interface data>	Render a new user interface on the IV (may cause the IH to launch an new IV).
Quit_Interface		Indicates that the user wishes to terminate the process.	*Hide_Interface*		Tell the IH to kill the current IV.
			Quit_Interface		Indicates that the PH wishes to terminate the user interface.

Table 1 : Typical Message Events between the IH and the PH

At the time of writing, the distributed user interface prototype had been successfully implemented on a PC running Windows NT and various flavours of UNIX machines running X-Windows. It is conceivable that this prototype model could be ported to other operating systems that support TCP/IP[1], an ANSI-compliant C compiler and a suitable IV for HTML (Netscape, Mosaic, Lynx, etc.). Work is being undertaken to investigate compatibility for TCL/TK scripts and VRML.

3. Conclusions

It has been shown that there is a great deal of work that needs to be undertaken in the field of distributed and heterogeneous user interfaces. The approach of designing the user interface code with the functionality code of a process is unsuitable and inflexible within a distributed system where there are potentially many user interface environments. The model that has been developed has shown how a distributed process can achieve user interface distribution, user interface abstraction and user interface remote control through the division of a process into three constituent components. The greatest single advantage of this approach is that it does not attempt to replace any individual user-independent

[1] The de facto networking protocol used throughout the Internet.

environment methodology or display language, but provides a complementary mechanism for their distribution.

User interface distribution is essential in allowing users to reuse network resources through distributing the processing and functionality across a distributed system, so long as user interfaces can be redirected locally to the user. This is advantageous, since it means that a user is not always forced to work at a particular display machine; they can work on the platform and within the user interface environment that is most suited to their working protocol. This is useful for people who work on multiple machines within a distributed system and prefer a consistent interface to their processes and to the way their data is presented.

User interface abstraction is required in providing heterogeneity across various platforms and user interface environments. The use of user interface independent display languages, although limited by the fact that they have to conform to common denominators and so cannot make full use of a user interface environment, do allow user interfaces to be constructed rapidly and rendered in a consistent manner. By supporting multiple user interface independent display languages, the model exhibits the properties of flexibility and extensibility in the types of data can be render and also the user interface representations can be supported.

User interface remote control is an area that was initially developed to provide a programming interface between the PH and the IH, and to allow PHs and IHs to control the user interfaces and functionality of other processes. However, there is great potential in this area for extending this programming interface to support mobile user interfaces and to allow distributed user interface processes to be controlled by agent technology, through a language such as the Knowledge and Query Manipulation Language [6]. This would allow users to program agents to reply to selected user interfaces on their behalf and also to communicate with PHs to collect and gather information or to perform updates for consistency control across user interfaces.

References

[1] BARTA, R. A. and HAUSWIRTH, M., Interface-Parasite Gateways. *In:* The World Wide Web Journal Issue One, Conference Proceedings of the Fourth International Word Wide Web Conference, Boston, USA, O'Reilly and Associates Inc., pages 277-290, 1995.

[2] BELL, G., PARISI, A. and PESCE, M., Virtual Reality Modelling Language Specification 1.0, 1995.

[3] BERNERS-LEE, T. and CONOLLY, D., Hypertext Mark-up Language Specification 2.0, 1995.

[4] BERNERS-LEE, T., FIELDING, R. and FRYSTYK, H., Hypertext Transport Protocol Specification 1.0 Internet draft 04, 1995.

[5] DAVIS, H. C., HALL, W., HEATH, I., HILL, G. J. and WILKINS, R. J., Towards an Integrated Information Environment with Open Hypermedia Systems. *In*: D. Lucarella, J. Nanard, M. Nanard and P. Paolini, *Eds.* ECHT '92, Proceedings of the Fourth ACM Conference on Hypertext, Milan, Italy, ACM Press, pages 181-190, 1992.

[6] FININ, T., et al, Knowledge Query and Manipulation Language.

[7] GOOSE, S., The Design Of A Generic, Yet Customisable, Distributed Communication Framework, Multimedia Technical Report M95/4, Department of Electronics and Computer Science, University of Southampton, 1995.

[8] GOOSE, S., DALE, J., HILL, G. J., DeROURE, D. C. and HALL, W., Unifying Distributed Processing and Open Hypertext through a Heterogeneous Communication Model, Multimedia Technical Report M95/6, Department of Electronics and Computer Science, University of Southampton, 1995.

[9] GOSLING, J. and McGINTON, H., The Java Language Environment: A White Paper, 1995.

[10] Netscape Communications Corporation, The Netscape Navigator, 1995.

[11] NYE, A., Xlib Programming Manual Volume 1 (3rd edition), O'Reilly & Associates Inc., 1993.

[12] OUSTERHOUT, J. K., TCL and the TK Toolkit, Addison-Wesley Publishing, 1994.