A Generic Model for Integrated Multi-Channel Information Systems

Richard Cooper and XianQing Wu
Computing Science, University of Glasgow, 17 Lilybank Gardens, Glasgow G12 8QQ
rich@dcs.gla.ac.uk

Abstract. There is an increasing need to provide methods for describing information systems which incorporate the integrated use of multiple channels of interaction. To achieve this effectively, it will be important to model such systems in a way which embraces the very many aspects involved – the data being used, the activities which are to be carried out and the user interfaces specific to each channel. This paper describes a model in which all of this can be described as a set of interacting parameterised components drawn from an object-oriented hierarchy. In the model, data sources, views, activities, channel dialogue models, forms, tables, menus, text, numbers and multimedia files are among the components which can be built up independently. The paper describes the model and discusses examples of its use and reports on the early stages of implementing a software environment for using the model.

1 Introduction

The number of ways of providing fast distributed access to important information is expanding quickly. Starting with the Internet and the World Wide Web, but continuing with interactive TV, telephone systems and the expanding capabilities of mobile devices, there are now many ways in which we can gain remote access to, for instance, financial, commercial and entertainment information. Companies wishing to provide services for customers and organisations wishing to distribute and solicit information have a wealth of possibilities. Building an information system which can only be deployed over the web is a less and less useful achievement. Unfortunately, each of these possibilities for information access (called channels from now on) is usually achieved with a distinct programming effort and tends to occur separately, at most sharing a database for the information content, but often not even sharing that. What is required is the ability to create Multi-Channel Distributed Information Systems (MCDIS) in which user access is integrated whatever the channel.

The proper place to start is to build the system anticipating multi-channel use from the outset. The builder of the information system should be able to construct the system, including each interface, as an integrated whole, sharing not only the "data", but also the descriptions of the processes involved. For instance, to order goods using a credit card requires the submission of the same information whether this done by typing at a keyboard, using voice input or using a phone pad. Therefore, the process of
eliciting credit card information should be described firstly in a device independent way and later specialised to whichever channels are required.

The development of an MCDIS would best be achieved using an environment which encompasses the ability to create device independent designs as well as the later deployment to specific technologies for use with specific channels. Such an environment would be built around a complete and coherent model which includes the ability to describe both abstractly and concretely all of the components of such a system. An object-oriented component-based approach seems most appropriate here since this can best provide the power to describe aspects of the system at different levels of abstraction, thus permitting the integration of all aspects of the system design.

So what are the aspects that have to be specified? Firstly, there is the nature of the information being managed. Then there are activities to be carried out with this information, which will mostly concern information capture and dissemination. Lastly, there is the human interaction required to accomplish the activities. This will involve mapping abstract descriptions of the activities onto the specific capabilities of the channels. Thus it is necessary to describe data sources, the data accesses, the activities which involve those data accesses, the channel-specific overall structures of interaction and the detailed fragments of interaction which fit into that overall structure.

A model in which everything is described as parameterised components allows us to describe each of these at a variety of levels of abstraction. If the model is supplied with abstract versions of every kind of component, these can be specialised to create specific information systems built out of concrete specialisations of the generic components. moreover the component-based approach permits the imposition of structures which support better software engineering practice than has traditionally be found when building such systems, including support for documentation, re-use and maintenance.

This paper describes such a model and how it can be used to build a MCDIS. The next section looks at other work in this area, before the model is presented in the third section. A methodology for using the model and example follow, before the paper ends with a section describing the current status and on-going work.

2 Related Work

One start point for research into MCDIS was the various attempts to bring discipline to web site construction. Content Management Systems (CMS) provide not only tools for the management of the data behind web sites, but also improved interfaces for the creation of the look and the structure of web pages [e.g. 1].

Whereas CMS simplify the implementation of web sites, Declarative Web Site Design Systems lift the focus to that of site design [e.g. 2, 3, 4]. Providing tools to capture high level specifications and separating the specification of site structure, content and formatting, enables the site designer to maintain a clear and maintainable description of the site from which the site can be generated.

WebML [5] is the most sophisticated of all of the Web Site design systems, the closest to the approach taken here. It provides the integrated description of web sites
including the data, site structure and page layout. It takes a component-based approach and provides useful commonly occurring components for each aspect of the site. It also supports the modelling of personalising information, which is already useful to tailor a web site to different tastes, but becomes much more useful when WebML is generalised to multi-channel use [6].

As attention is moved to other channels, some of the techniques described above can be used again. The interfaces to mobile phones bear a clear similarity to web sites to the extent that special versions of browsers are used. Voice and keypad telephone systems and interactive television also share similar features. They permit the user a fairly restricted method of interacting with the system mouse, keypad or restricted voice capability – and the stages of the interaction are typically described in an XML-based language.

There are, however, two important differences between channels - the capability of interaction provided and whether the temporal or spatial aspect of the interaction predominates. Channels vary in the quality of screen available, the input devices and the length of available battery-based power. A web site, using hyperlinks, provides a time-independent manner of access, while the stages of a telephone system must occur in a fixed order. Of course, for some activities, there is a temporal order whatever the channel.

Each kind of channel comes with its own XML-based description languages – indeed sometimes there are competitors for a particular kind of device. XHTML is the standard for web pages, DTMF and VoiceXML for telephone systems and WAP for mobile devices. However, it is not sufficient just to consider the languages which can statically describe interaction, we must also consider the dynamic aspect. Thus the description of dynamic systems must include a mechanism to describe dynamic components which rely on code to generate pages. Thus client side scripts and server side technologies such as PHP, JSP and so on will need to be describable. This also means that due attention be given the data source underlying dynamic systems and this should be integrated with the other aspects.

Therefore, when models are proposed to support multi-channel delivery, a more sophisticated architecture is required. Support systems now need to be able to deal with a more varied approach to the user interface, including support for both spatial layout and temporal sequences. They must also provide a high level view of the tasks which are to be achieved and they must still integrate these with a model of the information being managed.

The important issue becomes one of representing multiple abstraction layers to cope with the multiple devices. Teresa [7], for instance, supports descriptions of an abstract task model, platform specific task models, an abstract user interface and ultimately the concrete user interface. This work concentrates on the user interface as does RIHM [8], which provides a clean and device independent description of the layout. WebML is clearly already in a position to be expanded to cope with multi-channel systems [6]. In WebML, the personalisation mechanisms model different requirements while the hypertext model is extended to be context-aware. The Cottbus group concentrate on the detailed modelling of the tasks [9, 10] using a storyboarding approach. The Globis group keep the information modelling and demonstrate how the database can guide context dependence [11, 12].
Summarising this work, it is important to be able to describe data, task and user interface aspects in a way which is integrated, flexible and able to capture multiple levels of abstraction. We therefore propose a model in which all aspects of the MCDIS are modelled as components which take part in an object-oriented hierarchy. At any time, system designers can add new components by specialising pre-existing ones. Specialising now may be due to device specificity, personalisation or switching to new technology. The next section describes the model in more depth.

3 The Model

We now develop a model which encompasses the whole of a multi-channel information system. The model describes the MCDIS in terms of the data used, the activities to be carried out and the channels which are to be used. All are described using a component language which is common to all of the sub-models. Firstly the overall model will be discussed followed by short descriptions of the sub-model.

3.1 The Component Structure

A component is described in terms of a set of parameters which are given values in order to complete the specification. Each component, for instance, has parameters which hold basic metadata such as its name, a description, a date of creation, an author and its parent component – i.e. the one from which it has been derived. Components other than the most general component will have additional parameters defined on them. For instance, the main parameters of an MCDIS component are the data source, set of activities and set of channels used by that system, while a web page has a title and an associated style sheet, among other parameters, while a table has a set of column headings.

Each parameter has a name, a type and a set of constraints placed on it. The value of the type parameter is either the name of a base type (string or number) or is the name of an abstract component type. The constraints which can be imposed on a parameter include whether it is single or multi-valued and whether it is mandatory or not.

At the outset, the system provides generic components for all of the main kinds of component described below. Working with the model consists of specialising these by specifying parameter values or by adding new parameters. Therefore, it is possible to maintain partially specialised components for later re-use. For instance, a page style and layout can be created for re-use over a number of different information systems built for the same company. A component is concrete and therefore deployable if all of the mandatory parameters are set for all of the components which are reachable from it. An MCDIS component is the starting point of the description and when this is concrete, the whole information system can be deployed.
3.2 The Components

The structure of the MCDIS model is defined in the set of top level abstract components. These are as follows

1. MCDIS components consists of a data source, a set of activities and a set of single channel distributed information systems.
2. Although a data source could consist of several files or databases, it is describable as a single component in the form of a high level schema description [13] and a set of view or updates allowable from the information system. Specialisations of the abstract data source component are available for particular database systems and also for XML, which provide connection parameters. When these are instantiated, a concrete data source is available for deployment.
3. View and update components describe the mechanisms for retrieving and updating information in the data source. Abstract view and update components hold queries in an OQL-like language. Concrete components specialise these to implementation language such as SQL or XSLT.
4. An activity component describes a task to be achieved using the information system. Activities are described as having a purpose and as consisting of a collection of steps to be taken, similar to a workflow. The steps include instructions to the user, capture of user actions, feedback of information and the views and updates on the data source.
5. A single channel distributed information system (SCDIS) realises a subset of those activities. An SCDIS component has parameters for the set of activities, a dialogue meta-model, a label parameter (for instance the URL of a web site) and a starting point for the interaction, for instance the front page of a web site or the initial message in a telephone system.
6. Each channel type has an abstract dialogue meta-model which describes two things. Firstly, it describes the set of activities as a whole and how they are grouped – for instance in a menu or a table of contents. Secondly, it has the method of implementing the steps of a single activity. For instance, the meta-model consists of a set of hyperlinked pages for web sites and dialogue sequences for telephone systems.
7. The individual interaction points reachable the first interaction point either by user actions or temporally are called, for want of a suitable generic term, “pages”. There is a generic page component and specialised abstract versions for each of the different channels.
8. Each page will consist of a number of fragments arranged temporally or spatially. The fragments are hierarchically organised and are ultimately composed of fundamental data, mostly text strings and numbers, but also including multimedia data objects. There are therefore abstract components for each different kind of interaction objects, including the element types of XHTML, WML and VoiceML. These components are divided into blocks and in-lines, following the nomenclature of XHTML. Block components include tables and forms, while in-line components are the fundamental textual, numeric and multimedia objects which form the basis of the communication with the user. There are also a variety of style and layout components.
9. Some of the components are static consisting of explicitly specified sub-components. Other fragments are dynamic in the sense that they interact with the data source through the available views and update. There are therefore, dynamic versions of the abstract block and inline components. For instance, there is a dynamic table component which has a view parameter.

10. All of the above are describable using a single object-oriented parameterised component model and are thus available for re-use.

Fig. 1 The Architecture for an MCDIS

3.3 Discussion

The main values of this model would seem to be firstly that design can proceed independently of technological solution, multiple software vendors can place their products in a structure in which they can be easily integrated and that sets of generic components can be bundled up for customisation for specific purposes. Designers using the model also have the advantage of maintaining partially specified components – for instance, a generic page with a fixed style and layout can be created which is later specialised to each of the pages on the site.

Figure 1 shows an example of an MCDIS with a website and telephone channel. The MCDIS is parameterised on its activities, data source and channels. There are three activities and three views used. The diagram shows how activity A3 makes use of view V1 in one of its steps. The two channels have different dialogue models, but both have a start point as a parameter. The front page of the web site contains the table of contents, while the initial menu of the telephone system is a spoken menu. In both cases they lead to a subset of the activities, A1 and A3 can be achieved on the web site, while A1 and A2 can be achieved in the telephone system. The heavy use of inheritance is shown here only in that the specific activity implementations specialise
the generalised activities, but each of the pages and page sub-components will inherit from more general purpose versions.

4 A Methodology for the Model

The process of creating an integrated MCDIS consists of a number of steps and proceeds as follows.

1. Create the MCDIS object and label it with whatever documentary information seems useful.
2. Create the data source, setting the MCDIS parameter. At this stage this might merely mean creating the schema as an abstract data source, it might mean fixing the implementation system to be used and may even mean creating the concrete database with all of the connection parameters.
3. Specify the views and updates to be used.
4. Start creating the activities, describing their purpose first of all and then using the activity model to describe the steps. The views are available for use in the steps. Set the activities parameter in the MCDIS object.
5. Select one or more channels for use and set the MCDIS parameter to this set.
6. For each channel, identify the activities to be accomplished using this channel.
7. Map the set of activities onto the dialogue model for each of the channels.
8. For each activity, specialise the page model for that activity. Each step in the activity has to be associated with a page in the interaction model.
9. Define the interaction model. At this point, we are working with the components associated with specific technologies, e.g. XHTML pages and so on.

Firstly, a data source is specified. The description created must describe all of the data which is to be stored and the ways in which it is to be accessed. The description is at a high level and so is amenable to incorporation in the description of activities. The creation of the description may start from a pre-existing abstract component, for instance an ontology. Thus a bank accounting system would not be expected to start from scratch but from a bundle of components found useful for banking systems in general. In this case, the specialisation process may select from a large number of possible data types to store and may add others. It will pick particular views and updates and again may add others not found in the generic package.

The implementation of the data source will then consist of picking one or more databases or file structures and creating the mappings to the specific technology and adding extra information such as connection parameters.

Next the activities of the system can be identified. Again this process may start from the activities found in a generic bundle, but will usually include specification of new activities. The specification is at a high level and uses a workflow model which will discuss the generalities of the user interaction, page flow and so on.

Then the set of channels to be used will be selected and, for each, the activities to be realised over that channel. It is at this point that restrictions due to the channel capability are taken into account. There will later be the possibility of a check since in selecting specific components to implement the activities, it may be found that no
component exists for that activity over that channel, due to constraints on device capability.

Now the detailed structure of the channel information systems must be specified. Two strategies are available for this, either to create firstly an umbrella structure and specialise this for each channel or to work on each channel independently. Even in the latter case, the component structure means that a component identified when developing one channel may be re-used in another. This is the approach that is proposed here. The same text can be used globally for instance.

Continuing with the top down methodology, the page structure is described starting from channel specific abstract structures. A web site will be developed as a graph of web pages, for instance, while a telephony system will form a sequence of instruction-response pages. Each page can now be developed using whatever editor seems appropriate for the task. Thus a WYSIWYG graphical editor may be employed for web pages. However even in this case, the fragments of the page will be stored as separate components for later re-use.

![Fig. 2 A Simple Data Source Schema](image)

5 An Example

We take as an example a sports service for a football league. The service will include a web site, a mobile phone version and a touchtone telephone results service. The basis is a database whose schema is shown in Figure 2 and which holds information about clubs, players and matches. This schema is a specialisation of a much richer schema capable of holding data about team or solo sports, cups, leagues and umbrella organisations.

The database is described in terms of the high level model and a MySQL database is constructed to implement this. A concrete data source component is created to describe connection to the data source. The set of views are implemented in MySQL and then used to set the views parameter of the data source component.

The activities will include display the league table, display the results for one club, display the results for one day, accept a result, accept a correction and display a result matrix. Each of these is described by specialising the generic activity component. For
instance, the display the results for a particular day will involve steps requesting a date, retrieving that day’s results and displaying those results. The view retrieving results for a day will be used to describe one of those steps.

The creation of the web site starts with the identification of a style and a page layout. Creating the layout might, for instance, involve fixing a banner pane at the top holding an image, a table of contents pane (toc) on the left, an advertisement pane on the right and a content pane in the centre. The static content of the front page is then created and a link to the front page is added to the table of contents. The dynamic day results and club results pages are added next by specialising dynamic table, followed by a page listing the clubs, perhaps alphabetically and geographically using an image map. The club list page is added to the toc. The league table page is created with a content that again is mostly taken up by a dynamic table. It has hyperlinks to the club pages and is also is added to the toc. The results matrix page is added next and is also added to the toc. Finally a password protected result input page is added, whose content is mostly a dynamically generated form. When all of the components are fixed, the set of XHTML and PHP pages are generated for installation into the web site.

For a mobile phone, stripped down versions of the facilities can be given. The initial page (in this case a WML card) is merely a menu which leads to further cards for each of the activities. Note that many of the components generated for the web site can be re-used.

The telephone results system works similarly. The initial menu is now spoken with the user choosing between a set of numbers which could be requested using a VoiceXML <prompt> tag or may be better achieved using sound files. The options lead through a sequence to spoken results with identical content. Figure 3 shows the whole process.

![Fig. 3 Part of the Component Model for a Sports MCDIS](image-url)
In the figure, we see how an MCDIS is made up a data source, a set of activities and a set of SCDIS. The way in which activity components depend on view components is shown. (As in Figure 1, the single arrows indicate a part of relationship while a double arrow indicates an inheritance relationship.) The web site component is then shown with its URL component and its front page. This has a title and its body is made up of a table with a block of introductory text and the table of contents. This is a specialisation of an abstract menu of some or all of the activities which in turn is a specialisation of an abstract menu component. The telephone system is also shown with its first menu a different specialisation of the activity menu. The day results activity is shown (minus the date request) as an abstract activity implemented as an abstract dynamic table and implemented as a web page and a phone page.

6 Conclusions

We have described a component-based model (more fully described in [14]) for describing the whole of a multi-channel information system in such a way that the descriptions can be maintained and the implementation can be generated to suit a variety of technologies. The model is complete for the data source and single channel parts, but the precise formulation of the activity model needs firming up.

A development environment is being built which allows components to be created and edited and information systems to be generated. At present a subset of the component types are available as well as deployment to the web.

A reverse engineering utility has also been prototyped and promises the ability to take a well designed web site and recover the abstract components which underlie it. This would be extremely useful for enterprises which already have produced a quality web presence using an unstructured methodology and who now wish to develop their system in a more methodical manner and branch into other channels.

Acknowledgements

The authors would like to thank Philip Webb, Sakvinder Rai, Paul Marsland, Dominic Cardwell and James Constable for work on parts of the software.

References