Extending a Conceptual Modelling Approach to Web Application Design

Jaime Gómez[1], Cristina Cachero[1,⋆], and Oscar Pastor[2]

[1] Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante. SPAIN
{jgomez,ccachero}@dlsi.ua.es

[2] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia. SPAIN
opastor@dsic.upv.es

Abstract. This article presents OO-$\mathcal{H}$Method, an extension of the OO-Method conceptual modelling approach to address the particulars associated with the design of web interfaces. It is based on the OO-Method class diagram, which captures the statics of the system. The design of the interface appearance and the navigation paths are driven by the user navigation requirements. To achieve its goal, OO-$\mathcal{H}$Method adds several navigation and interface constructs to the OO-Method conceptual model, which define the semantics suitable for capturing the specific functionality of web application interfaces. A new kind of diagram, the 'Navigation Access Diagram' (NAD) is introduced. All the concepts represented in the NAD are stored in a repository, and from there a functional interface is generated in an automated way. One of the main contributions of this paper is not the proposal of yet another method for web modelling but the extension of an existing conceptual modelling approach.

1 Introduction

In the last few years the scientific community has conducted major research both on the desirable characteristics of hypermedia applications and on the concepts and processes that make up an effective environment for the structured development of such applications. Many of these concepts have been proven useful and therefore applied to a number of design methods, such as HDM [9], HDM-lite [7], OOHDM [19], RMM [11], ADM [1,12] or Strudel [6]. This article presents OO-$\mathcal{H}$Method, an extension of a conceptual modelling approach known as OO-Method that supports the conceptual design of web applications. This method [14,15] is a powerful proposal for software production from conceptual models. OO-Method is based on a formal object-oriented model OASIS [13] and its main feature is that developers' efforts are focused on the conceptual modelling phase. In this phase, system requirements are captured according to a predefined, finite set of conceptual modelling patterns (representation of relevant concepts at

⋆ This article has been written with the sponsorship of the Conselleria de Cultura, Educació i Ciència de la Comunitat Valenciana

© Springer-Verlag Berlin Heidelberg 2000

the problem space level). The full OO-implementation is obtained in an automated way following an execution model (including structure and behaviour). This execution model establishes the corresponding mappings between conceptual model constructs and their representation in a particular software development environment. Due to the lack of space, a formal description of OO-Method is not included. Interested readers are referred to [14]. OO-$\mathcal{H}$Method extends OO-Method, which implies that it relies on the information and functionality already provided by OO-Method, e.g. the control over pre and postconditions or the mandatory/optional nature of the value of the attributes. OO-$\mathcal{H}$Method allows the designer to center on the concepts needed to define a web user interface compatible with previously generated OO-Method applications. A new diagram is defined at the conceptual level. This diagram has specific semantics and notational characteristics to cover the observation, navigation and presentation aspects proven useful in the design of web applications. Following the philosophy of OO-Method, OO-$\mathcal{H}$Method applies two concepts: that of filters, based on the dynamic logic [10], and that of navigation patterns. Both the conversion of those patterns to their corresponding default presentation structure and the subsequent mapping of this structure into a set of default interface implementation constructs make possible the generation of the web architecture in an automated way. The code generated constitutes a true web interface to execute existing OO-Method applications over internet/intranet environments. The remainder of the article is structured as follows: section 2 provides a brief description of OO-$\mathcal{H}$Method in the context of the OO-Method approach. Section 3 presents the OO-$\mathcal{H}$Method conceptual model and describes in detail, by means of an example, the navigation access diagram that is used to capture user requirements semantics. Section 4 introduces the OO-$\mathcal{H}$Method execution model and shows the web interface that is generated out of the information captured in the navigation access diagram. A comparison with other related work is presented in section 5. Section 6 presents the conclusions and further work.

2 OO-$\mathcal{H}$Method

OO-$\mathcal{H}$Method is a generic model for the semantic structure of web interfaces, and so it is centered in global activities (authoring in the large) [9], that is, in classes and structures, and not in the content of the information nodes (authoring in the small). It is integrated in OO-Method, and extends the set of graphical elements necessary to get the abstract interaction model of the user interface. It also captures the information which each type of user (agent) can access and the navigation paths from one information view to another. An interface execution model is also provided in order to determine the way the conceptual model is implemented in a given developmental environment. It defines how interface and application modules are to communicate with each other. The navigation model is captured by means of the Navigation Access Diagram (NAD). Each piece of information introduced in the NAD has a corresponding formal counterpart, and is stored in a system repository. From there a functional interface can be generated in an automated way. The interface is coded according to the software environment chosen by the designer both on the client and on the server side.

The rest of the paper discusses both the conceptual and execution models.

3 Conceptual Model: The Navigation Access Diagram

As stated above, the navigation model is captured by means of one or more NAD's. It is necessary to have at least one NAD for each user-type (agent-type) who is allowed to navigate through the system. The first step in the construction of a NAD is the filtering and enriching of the information (classes, services and attributes) provided by the class diagram that was previously captured in OO-Method during the conceptual modelling phase. Conceptual modelling in OO-Method collects the system properties using three complementary models: the object model, the dynamic model and the functional model. For the purpose of this paper we are interested in the object model. The object model is a graphical model where system classes, including attributes, services and relationships (aggregation and inheritance), are defined. Additionally, agent relationships are introduced to specify who can activate each class service (client/server relationship). As each type of agent has a different view of the system and can activate different services, each one needs their own NAD. The filtering of the information is based both on the agent relationships from the object model and on a previous analysis of the navigation requirements, which sets boundaries to the information views required for each agent. This filtering process additionally allows the designer to create simplified prototypes and versions of the system. Furthermore, NAD's belonging to different OO-Method models could be combined under a common interface, which gives the method the capability to construct the system as a collection of collaborating web sites. For a complete perspective of the approach presented here, a small example is going to be employed: the library system. As a basic explanation (for reasons of brevity), it is assumed that, as is usual in such a system, there are readers, books and loans relating a book to the reader who orders it. There is a restriction that forbids a reader to have at any time more than three books on loan. If a book is not available, or the reader has already reached his three book limit, the book can be reserved. The librarian can search for books both by title and by author. Readers might play the role of unreliable readers in the system if one or more of their return dates had expired. If a reader is considered unreliable s/he cannot borrow any book. The class diagram of this example is shown in Fig. 1.

Classes are represented as rectangles with three areas: the class name, the attributes and the services. Inheritance relationships are represented by using arrows to link classes. For instance, the arrow between 'reader' and 'unreliable reader' denotes that 'unreliable reader' is a specialization of 'reader'. Aggregation relationships are represented by using a diamond from a given component class to its composite class. The aggregation determines how many components

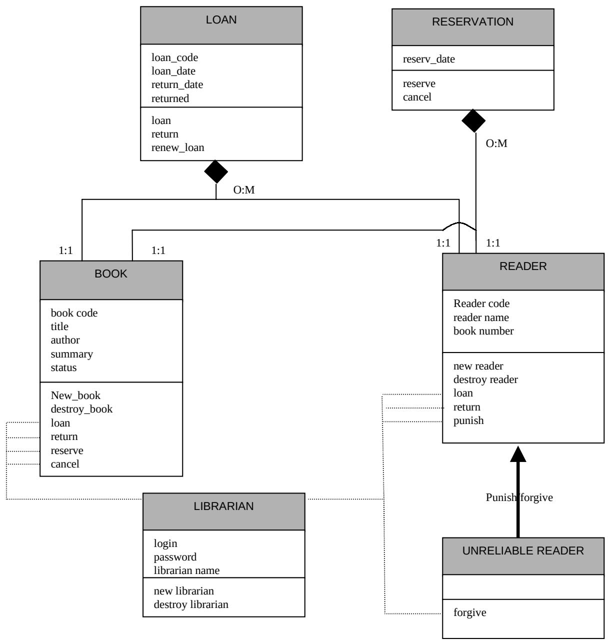


Fig. 1. *Class diagram of a library system*

can be attached to a given container and how many containers a component class can be associated with. For instance, the 'loan' composite class represents an aggregation between the component classes 'book' and 'reader'. Agent relationships are represented by using dotted lines that connect the associated client and server classes. In the example, the objects of the 'librarian' class can activate the services 'loan', 'return', 'reserve' and 'cancel' of the 'book' class. In order to design the NAD, the designer should know the navigation requirements of each type of user of the system. In the example it is assumed that the navigation requirements for the librarian user are as follows:

1. Lend a book. The librarian, acting on behalf of a reader, can search for a certain book either by author or by title. If the book is not available, or the reader already has three books on loan, a reservation on this book can be made for a future loan.
2. List active loans. For a given reader, the librarian can see at any moment the list of books s/he has borrowed from the library, and register the return of any of those books. Also, the librarian might want to look for extended information about each book (title, author, summary).
3. List out-of-date loans. The librarian can see the readers that have books with expired return dates, together with the date and the title of the books that should have already been returned. From this listing the librarian can decide whether to penalize them or not.

The main components of the NAD are navigation classes, navigation targets, navigation links and collections. Each of these constructs addresses the navigation model from a different dimension. We are going to further develop these concepts following the example presented above.

3.1 Navigation Classes

Navigation Classes (NC) have their grounding in the classes identified during the conceptual modelling phase, and are represented by means of a rectangle with three areas:

1. Head: it contains the name of the class
2. Attribute area: it contains the names and the scope of the attributes relevant to the considered agent and view.
3. Service area: it gathers the services available to the actual agent of the NAD.

All the attributes shown as part of a NC are accessible by the related agent, but the importance of each of these attributes may vary. OO-$\mathcal{H}$Method introduces the concept of attribute visibility in order to group the attributes depending on their relevance for the agent. This grouping will determine the way in which the user is able to access a given attribute. There are three types of attribute visibility:

1. Always Visible (V): their value is shown in every view of the object.
2. Referenced (R): their value is only referenced and so their consulting requires another step in the navigation path. The way of accessing this kind of attribute may vary depending on the implementation environment. In the case of working with HTML, for example, one possible way would be to show an anchor labelled as 'More Information'.
3. Hidden (H): Their value is neither referenced nor shown. The only way of accessing them is thus by means of a detailed view of the system.

This further differentiation of attributes reduces the possibility of the user feeling overwhelmed by the information shown on the views s/he has of the system.

Another relevant concept is that of Perspectives (P) [9,19]. The perspectives are defined on a set of edges, each one representing a relevant way of presenting the information. The perspectives types defined so far in OO-$\mathcal{H}$Method are:

1. Language: English, Spanish, German, French, Italian.
2. Rhetorical Styles: abbreviated, simplified, expert.
3. Medium: animation, image, text.

The best way of capturing perspectives in the NC is to define multivalued attributes, which are specified by means of a name and a set of values in brackets. Among the different perspectives, one of them must be declared as 'default', by means of a + sign. The perspectives have two possible scopes: global to the attribute type or local to each attribute. A global perspective is inherited by all the attributes of the type specified in the perspective. Global perspectives can be seen as 'visualization modes' [8], because they provide the application with different views of the same type of information to the same type of user. On the other hand, a local perspective adds possible visualization ways to local attributes. After deciding which information to show (navigation classes), the following phase is to define how to navigate through this information. When defining the navigation the designer must take into account many different aspects such as the order of objects, the filtering process or the cardinality of the access. These features are captured by means of different constructs associated with links, navigation targets and collections, which will be presented in the following sections.

3.2 Navigation Targets

The NC are grouped into Navigation Targets (NT). A NT is a set of NC which together provide the agent with a coherent view of the system. The general rule is to associate a NT to each user's navigation requirement. The NT has an associated scope: local to the actual type of agent (and so to the actual NAD) or global to the system. The graphical representation of an NT (see Fig. 2) is a rectangle that gathers all the classes involved in that view. An NT is defined by its name, which is located at a lapel in the upper-left corner of the rectangle. The concept underlying the NT is not new. Many authors have identified variants of such a concept: nodes [19], derived entities [9], hyperviews [7], macroentities [1] or targets[4]. Nevertheless, OO-$\mathcal{H}$Method uses a different approach, and bases its NT on user requirements, instead of on the physical presentation of the information (pages) as others do. For OO-$\mathcal{H}$Method, whether that information is presented in a single web page or divided among several pages is not important at this level of abstraction. In fact, the same NT could have several different materializations (pages), in a way similar to the 'targets' defined in [4].

The definition of the NT implicitly captures the 'navigation context pattern' [17]: the same nodes might appear in several NT as long as they do not represent the same information requirement, and in each NT its layout and navigation mode might be different. Note in Fig. 2 how, in the definition of 'book' provided in the 'loan book' NT, the principal attributes have been marked as 'always visible (V)' and two local perspectives have been added to the title of the book: text (default) and image (corresponding to a photo of the book cover).

3.3 Navigation Links

A Navigation Link (NL) is defined by:

1. Name.
2. Source navigation class.
3. Target navigation class.

4. Associated navigation patterns.
5. Associated navigation filters.

In OO-$\mathcal{H}$Method there are four types of NL:

1. Lr (requirement link): it shows the entry point to the NT. Every NT has a requirement link, which is drawn as a black circle with an arrow pointing at either the root navigation class or a collection inside that NT.
2. Ls (service link): it points at a service of a navigation class, and is drawn as a ray-arrow. It might have values of parameters associated.
3. Li (internal link): both its source and target NC remain inside a given NT. Its main characteristic is that its activation does not change the user context and, usually, it does not produce user disorientation.
4. Lt (traversal link): it is defined between navigation classes belonging to different navigation targets, and thus defines alternative visualization paths to the objects of the target classes.

The navigation links are always directed. This means that, if there is a need to navigate in both senses, two links must be explicitly or implicitly specified. The link label will be used as its identifier (for example, as the anchor text of the generated web page) so it should have a semantic meaning.

Furthermore, as shown in Fig. 2, all link types have both 'navigation patterns' and 'navigation filters' associated. Next both concepts will be developed.

Navigation patterns A navigation pattern is defined as a mechanism that allows a web user interface to share its knowledge about the way of showing the information objects to the user. Some authors [2] call them 'linking patterns'. OO-$\mathcal{H}$Method defines four navigation patterns, which can be associated both with navigation links and collections (see below). These are:

1. Index: access to a list of links to the different objects that form the population of the visualized class. In the view of each object there is always, added to the links derived from the semantic relationships among the classes being visualized (and captured in a navigation pattern), a link to the index page.
2. Guided Tour: it provides access to an object of the target population (depending on the associated filters) and a set of four operations: next, previous, first and last, in order to navigate through this target population.
3. Indexed Guided Tour: it combines the index with the guided tour mode.
4. Showall: It shows all the target objects together. It has an added attribute called 'cardinality' which allows the pagination of the answer and thus the limitation of the size of the pages and the load time for the user.

These patterns have been inherited from other well-known models such as HDM-lite [7], and enriched by adding a set of attributes that complete the navigation mode they represent. An example of this enrichment is the pattern attribute 'show in origin/show in destination'. The 'show in destination' attribute means that the information in the target objects will be shown in a different page, while 'show in origin' will present all the information about the target objects together with that of the source object. The selection of the most suitable navigation pattern depends on two variables:

1. Semantic relationships remaining under the link.
2. Granularity of the link: how many target objects are likely to be related to each object of the origin class.

Navigation Filters Associated to links and collections, a set of Navigation Filters (NF) can also be defined. A navigation filter restricts the order, the quantity (number) or the quality (characteristics) of the target objects. Formally, a navigation filter is a well formed formula (expressed in a subset of the dynamic logic [10]) that establishes a restriction on the attributes of the target class.

There are three types of filters:

1. Attribute filters: they specify values that must be conformed by the corresponding attribute values of the target population.
2. Condition filters: they can represent either method parameters (if associated to a service link) or additional rules and restrictions on the target population. A $ sign as the value of the filter means that such value has to be given by the user before traversing the corresponding link. This mechanism provides a means to define user-dependent target populations.
3. Order filters: they specify the order in which the target population will be accessed.

3.4 Collections

Another important decision about the user interface is how the information should be grouped in order to make it more accessible to the final user. In this context, another concept is introduced: the collection (see Fig. 2). A collection, represented by means of an inverted triangle, is a structure, hierarchical or not, which abstracts some concepts regarding both external and internal navigation. Collections have an associated scope (global, local to a NT or local to a NC), a set of filters and a set of navigation patterns associated and are a useful mechanism for limiting the interaction options between user and application, thus improving the usability of the system. A similar concept appears in different articles [6,4]. The collections can be nested but, as a rule of thumb, more than two nesting levels should be avoided, as they might disorient the user.

OO-$\mathcal{H}$Method defines four basic types of collections:

1. Classifiers: they define structures, hierarchical or not, for grouping information or service links which are to appear together.
2. Transactions: they group navigation services that are to work together. They should correspond with a transaction as defined in OO-Method.

3. Selectors: they group the objects that conform to the values gathered from the user. They usually have a condition filter associated.
4. History collections: OO-$\mathcal{H}$Method defines a history list [20] as a set of the last x links the user has traversed, where x is a parameter of this kind of collection and where the activation of one of these links will drive the user to the actual state of the required information, and not to the previous state, that is, the state it was in when it was visited for the first time.

Figure 2 shows a sample NAD of the librarian agent for the user navigation requirements specified above. Note that the access to the punish function is permitted inside the navigation target 'view out of date loans' which means that the librarian must check in this context the state of the reader's loans before punishing a reader.

4 Execution Model

The execution model provides the method with the representation details of the interface conceptual model for a target development environment. As the execution strategy is already defined in OO-Method, OO-$\mathcal{H}$Method is centered on defining how to implement the interface level information associated to web environments. All the concepts represented in the NAD are stored in an object repository and from there a default presentation diagram can be generated. Anyway, had the interface designer the need to refine such presentation and add some advanced features (e.g. multiple active views of the system), s/he could still edit this diagram (which will not be discussed here) and change it as needed. The subsequent mapping of this structure into a set of default interface implementation constructs provides the means to automatically generate the functional interface. The definition of the default navigation semantics (dependent on the semantic relationships existing in the conceptual schema) and that of a default interface layout (application of interface patterns to the elements of the conceptual schema) allow a CASE tool to suggest and/or derive certain application links and generate functional prototypes in an automated way. In Fig. 3 to 6 the prototype generated for the 'loan book' user navigation requirement is illustrated. The process is as follows: first, the generator tool looks for the entry point of the application. As nothing is stated in the diagram, it is assumed the entries to the different NT have an 'index' navigation pattern (see Fig. 3). Next, it generates a guided tour for the selected books. As the access to this class is made through a selector, the generator tool inserts a previous form where it asks for any of the fields (author or title) by which the search can be performed (see Fig. 4). The different screens of the guided tour (see Fig. 5) show all the information labelled as 'visible'. For each attribute perspective (expert summary, book cover image) a link is added to the form. In addition, the operations that can be performed on each object are also shown next to the book defining attribute (title). For each book there are two possibilities, depending on the state of the

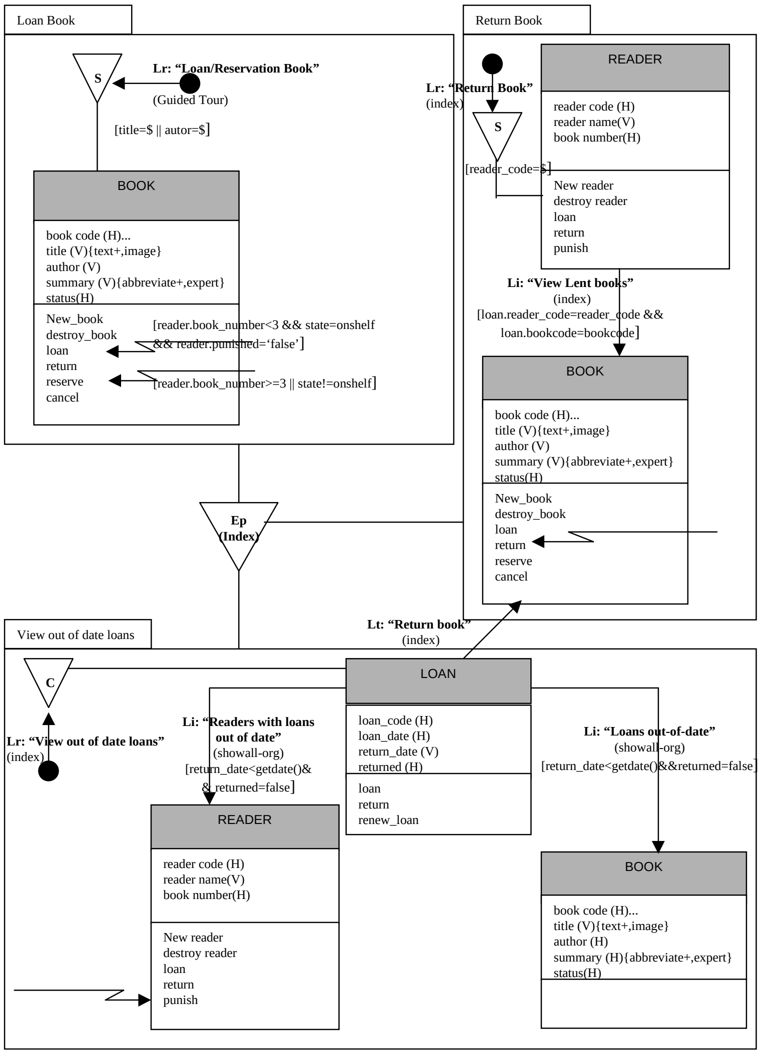
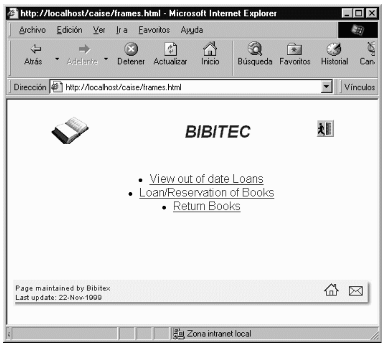


Fig. 2. *Simplified NAD of the Librarian Agent*

book and the number of books the reader has on loan. When the reader already has three books on loan or the book has been previously lent to another reader, then the service 'Make Reservation' is made available. On the contrary, when none of these conditions occur, then the available service is 'Lend Book'. As the value of the only parameter of the method 'Lend Book' (id-reader) must be introduced by the librarian, a form appears asking for its value. In Fig. 6 both this form and the system response is shown.

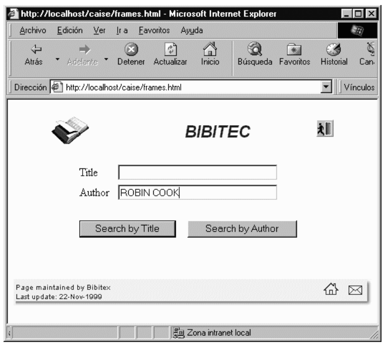

http://localhost/caise/frames.html - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial
Dirección http://localhost/caise/frames.html Vínculos
Zona intranet local

BIBITEC

- View out of date Loans
- Loan/Reservation of Books
- Return Books

Page maintained by Bibitex
Last update: 22-Nov-1999

Fig. 3. Library entry point

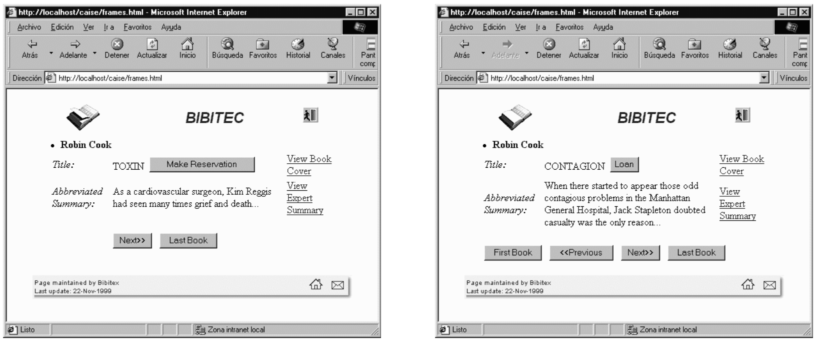

http://localhost/caise/frames.html - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial
Dirección http://localhost/caise/frames.html Vínculos
Search by Title
Search by Author
Zona intranet local

BIBITEC

Title
Author ROBIN COOK

Page maintained by Bibitex
Last update: 22-Nov-1999

Fig. 4. Book search


http://localhost/caise/frames.html - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Canales
Dirección http://localhost/caise/frames.html Vínculos
Make Reservation
Next>>
Last Book
Loan
First Book
<<Previous
Listo
Zona intranet local

BIBITEC

- Robin Cook

Title: TOXIN

Abbreviated Summary: As a cardiovascular surgeon, Kim Reggis had seen many times grief and death...

View Book Cover
View Expert Summary

Page maintained by Bibitex
Last update: 22-Nov-1999

BIBITEC

- Robin Cook

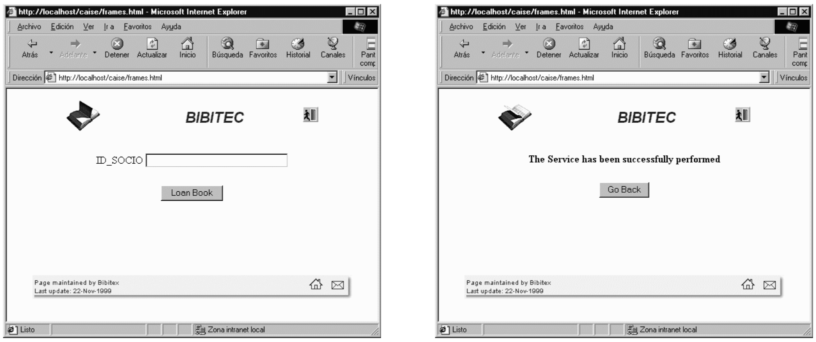
Title: CONTAGION

Abbreviated Summary: When there started to appear those odd contagious problems in the Manhattan General Hospital, Jack Stapleton doubted casualty was the only reason...

View Book Cover
View Expert Summary

Page maintained by Bibitex
Last update: 22-Nov-1999

Fig. 5. Guided tour applied to the search result


http://localhost/caise/frames.html - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Canales
Dirección http://localhost/caise/frames.html Vínculos
Loan Book
Go Back
Listo
Zona intranet local

BIBITEC

ID_SOCIO

Page maintained by Bibitex
Last update: 22-Nov-1999

BIBITEC

The Service has been successfully performed

Page maintained by Bibitex
Last update: 22-Nov-1999

Fig. 6. Loan *service parameters request and result page*

5 Comparison with Related Work

All the methods studied so far share many of the concepts proven useful in the design of hypermedia applications. As an example, it could be cited the concept of 'collection', whose basic meaning is captured in HDM by means of the outlines, in RMM by means of the grouping mechanism, in NCM by union nodes and aggregations, in OOHDM by context classes and in HDM-lite by collections. Another example is that of perspectives, which appear with the same name and meaning in models such as HDM, HDM-lite and OOHDM. OO-$\mathcal{H}$Method captures these and other concepts gathered in the classical hypertext theory, but has a number of features that makes it overall different from other models. From the OO-$\mathcal{H}$Method point of view, one of the main drawbacks of some of these models is that they are focused on the modelling of hypermedia systems, oriented to the information navigation and visualization [3]. In such systems there is no interaction with the user apart from their navigation decisions (or at least it is not taken into account when modelling the system). On the contrary, OO-$\mathcal{H}$Method extends the applications modelled with OO-Method, and so provides specific mechanisms for the user to interact with the system. The resulting hypermedia applications cover both the static structure and dynamic behaviour. Another important difference is that these models from the very beginning focus on the structural aspects in the solution space (the data and how it is going to be presented to the user), instead of centering on the structural aspects in the problem space. These solution-driven approaches lead to longer development periods and sometimes more complex design processes.

OO-$\mathcal{H}$Method aims at being simple: it tries to define intuitive diagrams instead of declarative or query languages, which are more likely to overwhelm designers without a strong computer science background. We also try to avoid restructuring information already captured in other models specially suitable for this purpose. That is the case of derived entities in HDM, nodes in OOHDM or macroentities in ADM. We believe that this information rearrangement mixes together the information and the navigation perspectives, which we try to keep as independent as possible. In our approach, navigation is captured by means of links and structures associated to them (patterns, filters and collections), while the relevant information is captured in the classes inherited from the conceptual level. OO-$\mathcal{H}$Method is explicitly a user-driven approach. By introducing the concept of 'navigation target' we are grouping the functionality of the interface in separate modules, possibly designed by different people, that aim at meeting a user requirement, in the sense of a specific functionality asked for by the client. OO-$\mathcal{H}$Method shares some apparent similarities with the OOHDM model [18]. This fact is partly due to the OO-approach both methods take. The advantages of the OO-approach are discussed in [19]. Both of them derive from a similar class-schema that models the problem domain. Also, both methods clearly separate conceptual design, navigation design, presentation design and implementation. However, OO-$\mathcal{H}$Method simplifies the process of defining each phase by a mapping mechanism that can be directly applied to the class diagram. In OOHDM attributes with multiple perspectives become different attributes of the 'node class'. In our OO-Model, we exploit the existence of multi-valued attributes, and so maintain the concept of 'one attribute for each concept' independently of how they are stored in the database. Also, determining that the links (that is, the way a user is going to navigate through the classes) are attributes of the nodes again is closer, from our point of view, to the database storage structure than to a navigation requirement. In the last months the use of UML for the modelling of web applications [5] has also been proposed. We believe that this extension is particularly useful once you already have both a conceptual design and a specific implementation platform. The reason is that, rather than abstract implementation constructs, the UML web extension introduces concepts such as ActiveX controls, components or forms as classes of the model.

6 Conclusions

Conventional object-oriented methods have to provide a well-defined software development process by which the community of software engineers can properly design web-based applications from requirements in a systematic way. Our purpose has been to address these problems in the context of a conceptual modelling approach that has been proven successful for software production from conceptual models. The OO-$\mathcal{H}$Method can be seen as an extension of OO-Method to face the whole software production process. We focus on how to properly capture the particulars associated to the design of web interfaces. In order to achieve this goal, OO-$\mathcal{H}$Method adds several navigation and interface constructs to the OO-Method conceptual model, which define the semantics suitable for capturing the specific functionality of web application interfaces. A new kind of diagram, the navigation access diagram, has been introduced. Each piece of information introduced in the NAD is stored in a system repository. From there a functional interface can be generated in an automated way. As oppose to other existing web methods, the approach presented in this paper does not intend to be 'yet another method' for web modelling but to extend a consolidated conceptual modelling approach.

Summarizing, the most relevant contributions of this paper are the following:

1. The detailed presentation of the OO-$\mathcal{H}$Method approach as a successful attempt to cover the entire web-based software production process from an OO point of view in order to get the best from conventional and formal methods.
2. The description of the NAD that has been added to the OO-Method conceptual model to specify the navigation user requirements.

OO-$\mathcal{H}$Method is still defining and cataloguing a set of both navigation and interface patterns general enough as to guarantee the code reusability. Navigation patterns also aim at providing the user with a higher level of usability, and the designer with a repository of well known and useful navigation techniques. The identification of interface patterns is a much more open process, as the new technologies will certainly add more effective ways of displaying the information in a given environment. The usability of the different patterns once implemented will be tested and will become a critical factor for its final incorporation in OO-$\mathcal{H}$Method. Once completely categorized, a formal specification of user interfaces [16] will follow directly from these interface patterns.

Acknowledgments

We would like to thank the anonymous referees for their valuable comments to this work

References

1. P. Atzeni, G. Mecca, and P. Merialdo. Design and Maintenance of Data-Intensive Web Sites. In *Advances in Database Technology - EDTB'98*, pages 436–449, 03 1998.
2. M. Bernstein. Patterns of Hypertext. In *HYPERTEXT '98. Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space. Structure in hypermedia systems*, pages 21–29, 1998.
3. M. Bieber and C. Kacmar. Designing Hypertext Support for Computational Applications. *CACM: Communications of the ACM*, 38(8):99 – 107, 1998.
4. S. Ceri, P. Fraternali, and S. Paraboschi. Design Principles for Data-Intensive Web Sites. *SIGMOD Record*, 28:84–89, 03 1999.
5. J. Conallen. Modeling Web Application Architectures with UML. *CACM: Communications of the ACM.*, 42(10):63–70, 10 1999.
6. F. M. Fern ndez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the Boat with Strudel: Experiences with a Web-Site Management System. In *Proceedings of ACM SIGMOD International conference on Management of data*, pages 414–425, 10 1998.
7. P. Fraternali and P. Paolini. A Conceptual Model and a Tool Environment for Developing more Scalable, Dynamic, and Customizable Web Applications. In *Advances in Database Technology - EDBT'98*, pages 421–435, 1998.
8. F. Garzotto, L. Mainetti, and P. Paolini. Designing Modal Hypermedia Applications. In *Proceedings of the eight ACM conference on HYPERTEXT '97*, 1997.
9. F. Garzotto and P. Paolini. HDM A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems (TOIS)*, 11(1):1–26, 01 1993.
10. D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic*, volume 165 of *Synthese Library*, chapter II.10, pages 497–604. D. Reidel Publishing Co., Dordrecht, 1984.
11. T. Isakowitz, E. A. Stohr, and V. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *CACM: Communications of the ACM.*, pages 34–44, 08 1995.
12. G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The ARANEUS Guide to Web-Site Development. Technical report, Universidad de Roma, 03 1999.

13. O. Pastor, F. Hayes, and S. Bear. OASIS: An Object-Oriented Specification Language. In P. Loucopoulos, editor, *Proceedings of CAiSE'92 International Conference*, volume 593 of *LNCS*, pages 348–363. Springer-Verlag, 1992.
14. O. Pastor, E. Insfr n, V. Pelechano, J. Romero, and J. Merseguer. OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In *CAiSE '97. International Conference on Advanced Information Systems*, pages 145–158, 1997.
15. O. Pastor, V. Pelechano, E. Insfr n, and J. G mez. From Object Oriented Conceptual Modeling to Automated Programming in Java. In *ER '98. International Conference on the Entity Relationship Approach*, pages 183–196, 1998.
16. S. R. Robinson and S. A. Roberts. Formalizing the Informational Content of Database User Interfaces. In *ER '98. International Conference on Conceptual Modeling*, volume 1507, pages 65–77. Springer, 11 1998.
17. G. Rossi, D. Schwabe, and A. Garrido. Design Reuse in Hypermedia Applications Development. In *Proceedings of the eight ACM conference on HYPERTEXT '97*, pages 57–66, 1997.
18. D. Schwabe and R. Almeida Pontes. A Method-based Web Application Development Environment. In *Position Paper, Web Engineering Workshop, WWW8*, 1999.
19. D. Schwabe, G. Rossi, and D. J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In *Proceedings of the the seventh ACM conference on HYPERTEXT '96*, page 166, 1996.
20. L. Tauscher and S. Greenberg. Revisitation patterns in World Wide Web navigation. In *CHI '97. Proceeding of the CHI 97 conference on Human factors in computing systems*, pages 399–406, 1997.