Advanced Conceptual Modeling of Web Applications: Embedding Operation Interfaces in Navigation Design

Cristina Cachero and Jaime Gómez

Web Engineering Group
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante. SPAIN
{ccachero,jgomez}@dlsi.ua.es

Abstract In the last years the integration of operation calls in the context of dynamic, personalized interfaces has become a 'must' for Web Applications. However, existing conceptual modeling proposals have just tangentially tackled the conceptual definition of its corresponding operation interfaces, which many times involve concepts far more complex than those related to text field input. This paper presents the OO-H modeling proposal for the seamless integration of both single and multi-step operation interfaces in the context of its interface navigational model. The notation and semantics provided by OO-H allow the reuse of context information and/or navigation paths for the assignment of values to operation parameters and the visualization of operation results.

1 Introduction

In the last few years we have witnessed how existing and yet to come web technologies have induced much more flexible distributed environments where new business opportunities have appeared, but also new risks related to software development [15]. Although the scientific community agrees in that, in order to keep the possibility of failure to a minimum, the development process for enterprise applications should evolve in a Web Engineering manner, there is no agreement at how the core activities behind a sound Web Application development process should be addressed, nor at how and to which degree the system functionality should be supported.

In this sense, some approaches, most of them coming from the hypermedia community, consider Web Applications as information delivery systems, and provide partial or full support to well known aspects such as navigation, presentation or personalization. However, and although in these proposals CRUD[1] operations are usually first-class citizens [14, 16, 3], they lack specific mechanisms to model the interface with more complex, general-purpose operation interfaces that are

[1] Create, Read, Update, Delete

required in current applications. Other approaches, coming from the Software Engineering field, regard Web Applications as traditional distributed applications, and propose operation modeling approaches that, making exclusive use of standard models and methods, capture the idiosyncrasy of the user-system interaction [4]. These approaches, very powerful, do not however provide the necessary level of abstraction from implementation issues such as target technologies and/or platforms. Therefore they fall short to cover topics such as conceptual navigation support, platform and language independent page architectures, personalization concerns etc., which are crucial aspects in Web Engineering. Finally, cost reduction and time-to-market speed of sophisticated web applications have caused a leverage effort of services[2] from partners and other third parties over Internet. This effort has been materialized in the inclusion of service integration activities as part of the web application development process [7, 9]. Approaches following this Dynamic Business Web trend consider that the application development should mainly consist of a process of communication and integration of Web Services disseminated over the net and offered via (often) collaborating technologies such as Universal Description, Discovery and Integration (UDDI [18]), Directory Services Mark-up Language (DSML [6]), Simple Object Access Protocol (SOAP [17]) or Web Services Description Language (WSDL [22]). This dynamic vision, which adheres to the new service-oriented paradigm, disregards however from our point of view the existing linkage in real applications between the operation interface view and other hypermedia perspectives (e.g. navigation, presentation, personalization) that make up the Web Application interface. We agree with [13] in that each of these trends partially addresses the nuances Web Applications involve, and that only a fusion of their respective points of view would provide a cohesive solution to its conceptual modeling.

This article follows this integrating philosophy and presents, in the context of the Object-Oriented Hypermedia method (OO-H [10, 11]) a set of conceptual models and constructs that enrich the traditional views offered by hypermedia methods (namely the domain, navigation and presentation view) to capture the necessary operation interface semantics. These new constructs provide the front door to the underlying business logic that is to be integrated in the application under development. One of the OO-H main contribution is an operation-aware navigation view of the application interface. This fact responds to the OO-H claim that user navigation is not necessarily restricted to relationship traversal among domain classes. Far from it, we agree with [2] in considering that operation invocation interfaces and, more precisely, (1) introduction of operation parameter values and (2) visualization of results, may require a user interaction far more complex than filling a form or reading a string informing on success or failure of the underlying method execution. It is precisely this complexity what makes advisable that operation interface models explicitly include user naviga-

[2] In this article, unless otherwise specifically stated, we will use the term *service* in a technological sense, to refer to an interface that grants the user access to a method (which constitutes the implementation of that service).

tion as a way to improve its usability.

The remainder of the article is structured as follows: first, section 2 describes an overview of the modeling phases of OO-H Method. This section presents, by means of a comprehensive example, the OO-H models and constructs relevant for the conceptual modeling of operation interfaces, and provides a basis for Section 3, where the different operation interface concepts are described in detail. Section 4 contextualizes our work and provides a brief overview of the treatment that other well-known hypermedia conceptual modeling approaches have provided to support operation interfaces. Section 5 describes final remarks. Last, conclusions and further work are presented in section 6.

2 OO-H Modeling Phases

The OO-H method is a generic approach, based on the Object Oriented paradigm, that provides the designer with the semantics and notation necessary for the development of web-based interfaces, which integrate both informational and operational views[3]. The OO-H modeling process is user-driven, what, for the sake of this article, implies that different actors may require different OO-H models in order to reflect their particular view of the application.

In order to illustrate the main OO-H constructs, a small example is going to be employed all along the paper: a *Hotel Reservation System*. In this system, and as a basic explanation (for reasons of brevity) let's assume we are interested in modeling the receptionist interface. Let's also suppose this actor has access to three hotel subsystems: *Client Management*, *Hotel Data Management* and *Booking Management*, each of which includes a subset of the user functional requirements. The receptionist is allowed, inside the boundaries of the *Client Management* subsystem, to add a new client, register the entrance of a client, add charges to rooms and print invoices. On the other hand, as part of the *Hotel Data Management* subsystem, s/he can manage the number, type and characteristics of the rooms, the service types offered to clients and the payment methods. Last, inside the *Booking Management* subsystem, the receptionist can view, add, update or delete client bookings. The first step in OO-H is to capture functional requirements in a UML-compliant [20] Use Case Diagram. For the purposes of this paper we are going to focus on the business Class Diagram that can be inferred from the use case specification.

2.1 Domain Modeling

This Business Class Diagram covers the definition of both static (classes, attributes and relationships) and dynamic (operations) aspects of the system.

[3] A whole definition of the OO-H approach is out of the scope of this paper; interested readers are referred to [10, 11] for an extensive explanation.

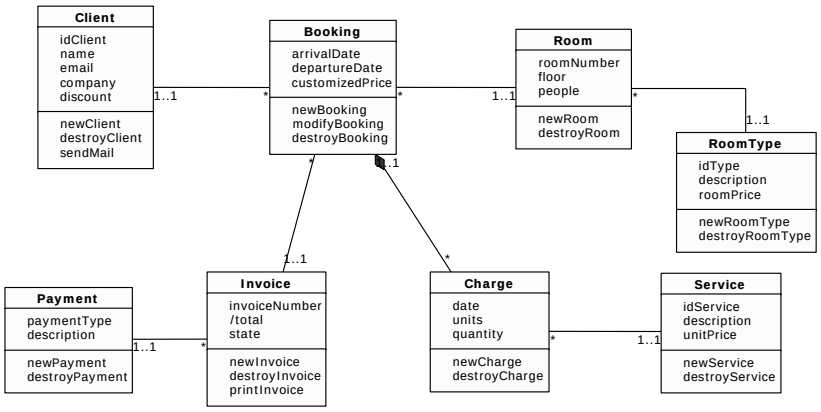

Figure1. *Hotel Reservation System Class Diagram*

In order to illustrate this phase, in Fig. 1 we observe the Class Diagram corresponding to the *Hotel Reservation System*. In this diagram each hotel is made up of a set of *Rooms* which can be of different *Types*. Rooms can be booked by *Clients*. For each *Booking* the system keeps track of the *Services* provided (laundry, drinks, etc), in order to *Charge* them to the client. On a client departure, an *Invoice*, which may include more than one *Booking*, is generated and the method of *Payment* is registered.

2.2 Navigation Modeling

Once the Domain Analysis has been completed, OO-H defines a design phase that involves, among other activities, the construction of a navigation view, materialized in a Navigation Access Diagram (NAD). Inside this diagram, and in order to manage the system navigation complexity, OO-H defines a package mechanism (known as Navigational Target, NT) that allows the designer to leverage the amount of information included in each NAD view. Although NT can be nested in any level, one of its most straightforward applications in OO-H can be seen in Fig. 2. There, we can observe how the *Hotel Management System* interface is organized around the three subsystems (*Client Management*, *Booking Management* and *Hotel Data Management*) identified during the analysis phase. Each subsystem has caused the definition of a namesake NT that encapsulates the navigation paths needed to fulfill the corresponding requirements specification. The different NT are accessed via a Collection (C), an access structure that is depicted as an inverted triangle and in this case models the entry point to each NT.

On the other hand, and already inside the boundaries each NT, the designer must define the (possibly restricted) views each user type has over the different conceptual classes. These views, known as Navigational Classes (NC), may include both attributes and operations, and must be connected by means of Navigational Links (NL), which provide the navigation paths that give a response to

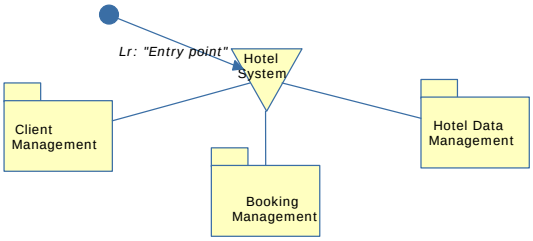

Figure2. *Level 0 NAD Diagram (derived from the Use Case Diagram)*

each user requirement. The notation for these constructs can be seen in Fig. 3, where the contents of the *Booking Management* NT (see Fig. 2) are presented. There, we can observe how the NC *View Bookings*, which is a restricted view on the Domain Class *Booking* depicted in Fig. 1, provides information about the arrival and departure dates and the price each client pays for each room booked. Also, this NT includes the definition of the *newBooking()* operation interface, which allows the receptionist to introduce new reservations in the system.

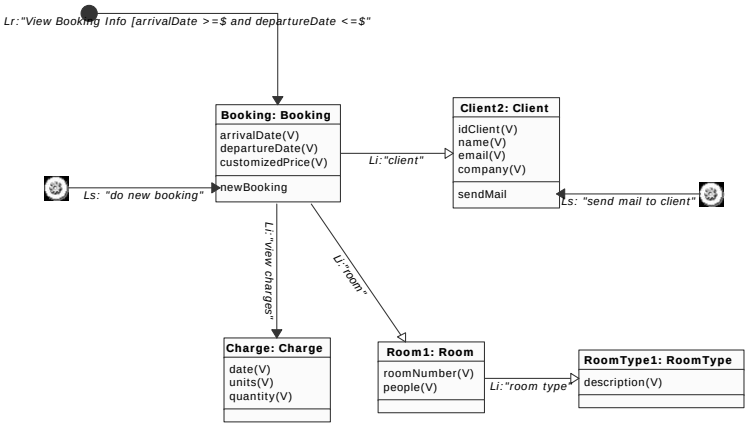

Figure3. *NAD corresponding to the Booking Management NT*

NL in OO-H are categorized according to its purpose. For the sake of our example, three types are especially relevant:

– Requirement Links (Lr) establish the point from where the user begins to navigate inside each NT. They are depicted by means of an arrow with a filled circle. In Fig. 3 the arrow labeled *View Booking Info*, which points at the *Booking Info* NC, belongs to this type.

– Internal Links (Li) connect information constructs inside a given NT. In Fig. 3 we observe how the *Booking* NC is the source of two Li (room and client) that relate the general information regarding each booking with information regarding the specific room and client involved in that booking instance.
– Last, Operation Links (Ls), whose detailed description will be provided in next section, model the interface to the operations that the actor is allowed to invoke. Operation Links are depicted by arrows with a serrated wheel. Back to our example, in Fig. 3 it can be observed how, inside the *Booking Management View*, the receptionist has been granted access to the *newBooking()* operation, which belongs to the *Booking Info* NC.

Also in our example we can observe how NL may have, among other characteristics, one or more *Filters* associated. Filters are OCL formulae [21] that constrain navigation. In order to illustrate this concept, the Filter *arrivalDate >= $ and departureDate <= $*, where the *$* symbol stands for user input, has been associated to the Lr *View Booking Info*. This filter restricts the set of target objects to those bookings that are inside the boundaries of a given period.

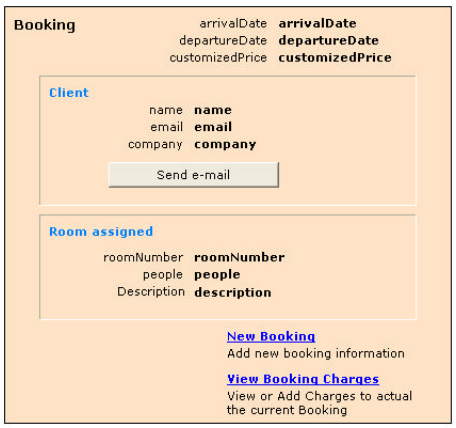

Figure4. Possible Storyboard for the *Booking Management* NT

It is important to stress the fact that the navigation design decisions influence the number and type of abstract pages that will finally make up the user interface[4]. The separation of attributes and operation calls among abstract pages is performed by means of a *Show-in* tagged value associated to links. This tagged value can be set to *origin* or to *destination*. NL with *Show-in=origin* (depicted by means of a hollow arrow head, see Fig. 3) gather in a single page the information contained in the origin and target NC. On the contrary, NL

[4] These pages can be later refined and integrated in physical views during the OO-H presentation design modeling phase, which falls out of the scope of this article.

with *Show-in=destination* (depicted as a filled arrow head, see Fig. 3) cause a new abstract page to appear in the interface. In order to further illustrate this fact, one pages making up the storyboard derived from Fig. 3 is presented in Fig. 4. The three *destination* NL (namely the Lr *View Booking Info*, the Li *View Charges* and the Ls *newBooking()*) give birth to three abstract pages. The first one see Fig. 4 shows a *Booking* list, with the associated relevant client and room information. This information has been defined as part of the Booking abstract page by defining two Li (room and client) of type *Show-in=origin*(see Fig 3). On the other hand, *destination* Li cause a set of anchors to appear on the page, namely those necessary to navigate either to the *Charges* or to the *newBooking()* page. Additionally, filters in OO-H implicitly define a new abstract page, that in this case allows the introduction of the dates of interest for the booking view.//

Inside the storyboard, the abstract page corresponding to the *newBooking()* operation is of special interest for the purpose of this article. The way this page is conceptually modeled is presented in next section.

3 Embedding Operation Interfaces in OO-H

As far as operations are concerned, the Class Diagram modeled during the Domain Analysis phase of the method may include information regarding aspects such as its *class* or *instance* scope, or whether they cause changes in the system state or not (*isQuery* tagged value). Also, the Class Diagram may include information regarding parameters involved in each operation, such as its type, whether they are input (in), output (out) or both input and output (in-out) parameters, mandatory/non mandatory character or even a default values if no specific value is provided. However, this specification does not cover information regarding the way the user may interact with such operations, that is, how s/he may enter values and view the results after its invocation. This gap is filled in OO-H by means of Operation Links.

Operation Links involve a set of parameters, to which OO-H associates a given *Introduction/Visualization Mode*. These parameters can be introduced in a single page or, on the contrary, be divided among different pages in what OO-H denominates *Multi-Step Operation Interfaces*. Also, and orthogonally to this fact, a given operation link may be associated to a single operation call (*Simple Operation Links*) or, on the contrary, it may imply several calls on the same or even different underlying method invocations (*Compound Operation Links*). All these concepts will be further developed in the following sections.

3.1 Parameter Introduction Modes

The first task the designer must perform in order to define an operation user interface is defining the way the parameter values must be introduced. In order to fulfill this task, each *Operation Link*, regardless of its type, inherits from the Business Class Diagram the set of parameters involved in the operation call.

These parameters are associated with an *Introduction Mode*, that define the way the user may provide values for them. OO-H distinguishes among five types of Introduction Modes:

- *Hidden parameters* take their value from a default OCL expression introduced by the designer, or a *null* value if such expression is left blank. They are not visible in the interface.
- *Constant parameters* only differ from hidden parameters in that their value appears in the interface, although without edit capabilities. These parameters usually provide relevant information for the user task.
- *Immediate parameters* adopt the value typed by the user at execution time. This value may be introduced by means of any kind of text input field.
- *Selection parameters* are given a value by means of a user selection among a set of predefined values. These values can be constant (extensionally specified by the designer during the modeling task) or context-dependent. In the second case the set of possible values is defined by means of an OCL formula, and the possible values will be calculated at execution time from the population of the system.
- Last, *User Navigation Parameters* imply a navigation activity through the model in order to get the desired value.

These *Interaction Modes* can be freely combined inside the boundaries of a given operation. Also, for different users, the interaction mode for a given parameter may vary. For example, let's suppose registered hotel customers are allowed to enter personal bookings. This fact implies that the *client* parameter associated to the *newBooking()* operation (inside the boundaries of the NAD corresponding to the Client view) should be set to *hidden* or *constant*, and adopt the value of the user identified in the system. It seems logical however that the receptionist is allowed to enter new bookings for any client, either new or previously registered. This fact is modeled in our example by associating to the *client* parameter a navigation path through the hotel client population, as we will show later.

Simple operations tend to present the parameter input process as a single step, that is, involving a single page where parameter values are introduced, either directly or after having performed any kind of navigation. However, more complex operations often require the modeling of different pages grouping related parameters. Think for example of the check-out process in an e-shop. In order to introduce the delivery data, the payment method, any kind of applicable discount etc. different interface pages, corresponding to different explicit steps, must be filled in. The user may go back and forth through them as many times as desired, and only when she presses the 'place order' button is the information sent to the back order system[5]. The way OO-H distinguishes among these two kinds of operation interfaces (*One-Step Operation Interfaces* and *Multi-Step Operation Interfaces*) is discussed next.

[5] Note how such 'place order' button represents a single operation from a user perspective, regardless of how it is implemented (e.g. by means of a transaction defined on several methods) in the underlying business logic module

3.2 One-Step Operation Interfaces

As mentioned above, the easiest operation interface configuration is based on a single interaction step. In order to illustrate this kind of interface, let's suppose the *newBooking()* operation requires the explicit introduction of four parameters: *arrivalDate*, *departureDate*, *Client* and *Room*. The *departureDate* and *arrivalDate*, due to their simple domain, may be modeled with an *introductionMode=immediate*, what causes a text box to appear in the corresponding storyboard (see Fig. 6).

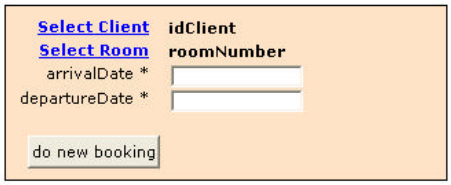

Figure5. *Possible storyboard for* new booking *operation*

However, in order to introduce the *Client* and *Room* involved in the booking instance, the user should be allowed to choose an instance from the population of the corresponding classes. As explained above, this fact is modeled by setting the *introductionMode=navigation*, and by defining a set of navigation paths to get the parameter values. Back to our example, imagine we have provided the application with *Client Search* and *Room Search* capabilities.

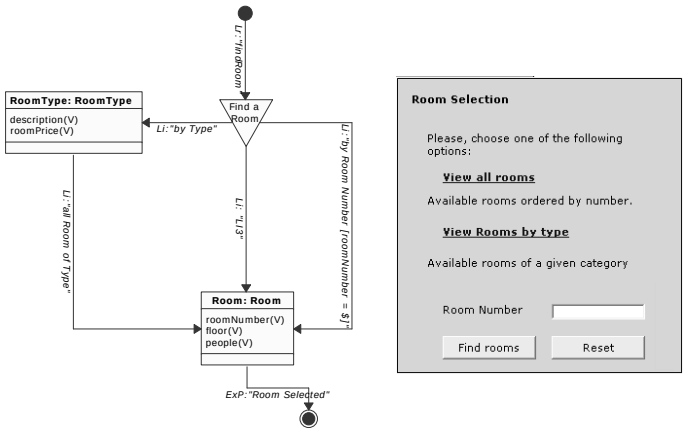

Figure6. *Possible storyboard for* find room *navigation*

Such capabilities are modeled in the NAD's presented in Figs. 6 and 7. The page snapshot corresponding to those NAD partial views are also shown in Figs. 6 and 7.

The association of one or more navigation paths to a given input parameter (such as *Client* or *Room* in our example) can be performed either in an intensional or extensional manner. On one hand the designer may establish any link as the initial or final link for a given parameter input path. Together, these links determine the subset of navigation paths the user may follow in order to enter a value (or a set of them) as part of the operation invocation. This technique is very useful when the designer is interested in reusing a whole navigation substructure. Back to our example (see Fig. 6), the designer may set the *Find Room* link as the initial link for the *Room* parameter, and *Room Selected* as the final link. In this way, all paths in between become available for the selection of the room, namely *Search by Type*, *Search by Room Number* and *Search all available rooms*.

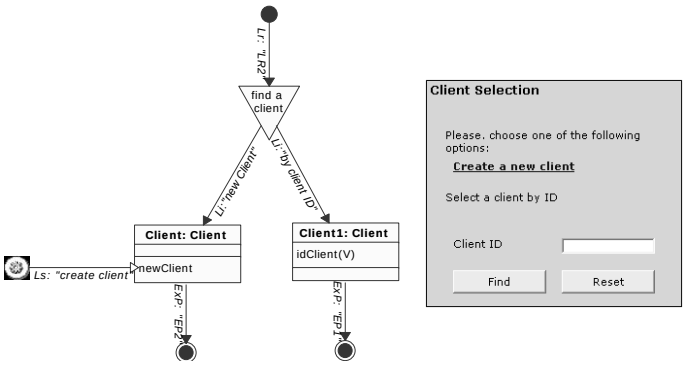

Figure7. *Possible storyboard for* search client *navigation*

All the concepts presented so far serve as a basis to conceptually model another very common situation in web operation interfaces: the division of the parameter input task into more than one *operation steps*. Next we are showing the OO-H modeling proposal for such situations.

3.3 Multi-Step Operation Interfaces

As stated above, Multi-Step interfaces involve the division of the parameters that must be introduced into a set of abstract pages, through which the user may go back and forth until s/he decides to activate the operation. In order to model such situation, OO-H allows the use of the Multi-Step Operation Link

modeling constructor. An example of such modeling can be observed in Fig. 8.

Multi-Step Interfaces are specially relevant when associated to *Compound Operation Links*. *Compound Operation Links* usually appear when interface usability concerns suggest the convenience of defining a common interface for a set of operations otherwise independent (that is, with no underlying transaction involved [6]). Among other advantages, *Compound Operation Links* allow the reuse of parameter values among method calls.

Imagine for example that we are dealing with a ticket reservation system where there is an operation called *newTicketReservation()*. Also imagine that we are interested in modeling an interface that allows, based on such operation, the purchase of not only one but a set of theater tickets for a given season. In this example it is clear that the fact that there are no tickets available for a given play does not imply that the other purchases need to be rolled back: we still may want to buy tickets for other plays. Furthermore, we will probably be interested in introducing our personal and payment data just once. An OO-H model for such situation can be observed in Fig. 8.

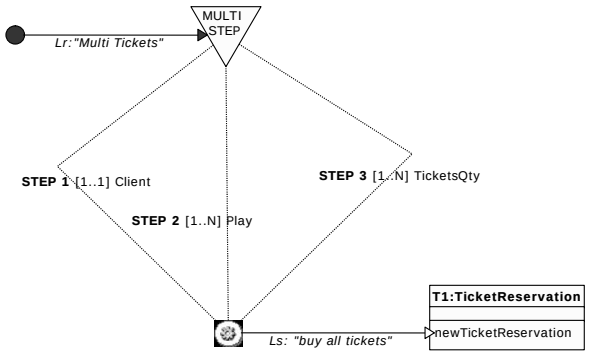

Figure8. *Multi-Step Operation Modeling*

Also in Fig 8 we observe how the parameter splitting process is modeled by means of different branches departing from the Collection *Multi Step*. Each step has an associated *cardinality*, that can be set either to *1* (meaning that only the last value introduced is stored and reused for each call) or *N* if the user is required to enter a different value for each iteration on the method. The latter case causes, at implementation time, that the application keeps track of the values introduced by the user and provides one them each time the method is invoked

[6] The transaction concept, although critical for the correct implementation of the application, falls out the scope of the OO-H interface model which, at the current stage of development, regards transactional behaviour as a business logic concern.

(what, furthermore, causes the 'consumption' of that value).

Back to our example (see Fig. 8) the multi-step interface consists of the following steps:

- In STEP 1 the user (client) must enter her personal data. The *[1..1]* cardinality means that this value will be reused among operation calls.
- In STEP 2 the user enters the set of desired plays. The *[1..N]* cardinality allows more than one play to be selected. The final number of plays will determine the number of iterations on the method *newTicketReservation()*.
- In STEP 3 the user introduces the number of tickets desired, whose value we have also decided to reuse among method calls. The user can go back and forth as many times as desired, until s/he decides every data is correct. Then, s/he activates the service, and gathers the results [7].

The set of constructs presented so far provide the necessary semantics and notation to model common situations in user-operation interaction. To the extent of our knowledge, there are no other efforts that tackle such aspect of Web Application interfaces at the same level of flexibility or with the same level of reuse. In the following section we present an overview of such related proposals.

4 Related work

The aim of OO-H has never been to become "yet another method" for Web Application analysis and design, but to integrate proven successful practices an concepts (use cases, links or collections to name just a few) and propose a set of enrichments to improve identified gaps. In this context, one of its most relevant contributions consists on the set of constructs and techniques it provides to model the integration of operation interfaces in the context of a sound navigation model. This aspect, due to the original perspective of Web Applications as information delivery systems mentioned in section 1, has been traditionally underestimated by the hypermedia community, which has mainly centered on navigation, information filtering and, more recently, on traditional database functionality (CRUD [8] operations).

ADM-2 [1] for example uses both a set of models and a declarative language to define not only navigation but also concepts such as transactions and activity workflow. One of its main characteristics regarding operation treatment is that it, similarly to OO-H, associates 'actions' to links. However, its database-oriented perspective limits the implication of such philosophy, and restricts it to an extended set of database-oriented operations. Another relevant conceptual modeling approach is HDM-2000 [8], whose strongest point is, from our

[7] Note that, without the specification of transaction control mechanisms, the completion of the method may involve the purchase of a subset of the selected tickets.

[8] Create, Read, Update and Delete

point of view, its treatment of the access structures to underlying information objects. Their multimedia orientation makes operation treatment a secondary aspect that is not explicitly tackled in the proposal. Also WebML [3] is a very intuitive approach that defines a set of 'Units' that capture a restricted set of operation types. WebML associates to these operation units a set of data entry units to collect input values. Another very well known proposal is OO-HDM [16], which, as far as operations are concerned, includes a 'Creation/Update' construct that, defined as part of its navigational diagram, also reflects underlying CRUD operations. One of the more interesting features of OO-HDM is the definition of a User Interaction Diagram that complements the navigational view. The user interface captured by means of this diagram, when associated to operation constructs, provides a powerful operation interface specification mechanism.

From our point of view, the inclusion of specific constructs for each operation type affect the flexibility of the methods adversely. Furthermore, the way of modeling such interfaces lacks, the level of navigation and conceptual reuse OO-H provides. Furthermore, all discussed methods implicitly assume simple parameter types, where only text field input is necessary to provide a meaningful value. Last, the kind of operations invoked by current Web Applications may be far more complex than CRUD operations tackled by the above presented database-oriented approaches. Although these proposals can be extended to deal with arbitrary business logic operations (see e.g. the 'General Operation Unit' in WebML), we claim that Object Oriented models, in which operations are first class concepts, provide a much more flexible platform to model web functionality. Specifically, the class diagram proposed in such methods provides information that may drive the construction of the operation interfaces, facilitating at the same time the functionality integration process. This object oriented premise is shared by other well-known proposals, such as UWE [12] or WSDM [5]. UWE supports Web Application development with special focus on systematization and personalization. Its UML-compliant notation for all the models adds precision to the notation, to the models and to the process. However, and although its object-oriented character allows the seamless inclusion of operation signatures in the domain model, it does not provide any clue (as far as we know) about how the introduction of parameters may be performed. Also WSDM follows an audience-driven, object-oriented approach. However, this proposal also faces the inclusion of operations in a very restricted way, by introducing a set of reserved words (NEW, REMOVE etc.) in its Functional Chunk (functional view of the system). Nevertheless, the common background these proposals share with OO-H would make easier, from our point of view, the inclusion of input mechanisms semantically similar to the ones proposed in this article.

5 Final Remarks

Finally, and once the Navigation Model has been defined, OO-H includes a presentation design model that, departing from such design navigation model, cap-

tures interface architecture and presentation. Although the definition of such model is out of the scope of this article, it is important to stress that this interface description is formalized in a taxonomy of XML templates, where each template gives a different interface perspective. The result is a graph of XML abstract pages whose interrelations are showed in a diagram called Abstract Presentation Diagram (APD) that depicts the interface *siteview*.

It is also important to stress the fact that, although the navigation design decisions influence the number and type of abstract pages that will finally make up the user interface, they are not still concerned with platform or language-dependent constructs. In that sense we can draw a parallelism between the APD level of abstraction and that provided by some well-known standards such as WSUI [23] or, at a lower level of abstraction, UIML [19]. In fact default WSUI specifications (with defaults set for action courses and error handling) could be derived from the NAD diagrams, while the OO-H presentation design activity completes the minimum set of information needed to generate default UIML interface specifications. The main advantage of the NAD over these specifications is, on one hand, its graphic nature, which makes it easier to use, and, on the other hand, the degree of reuse that its higher level of abstraction and the fact that it is based on other domain models (namely Use Case Diagram and Class Diagram) propitiates.

6 Conclusions and further work

This paper has presented an extension to the OO-H conceptual modeling approach for the specification of user-operation interaction (feeding of parameters, invocation of operations, either simple or compound, and view of operation results). This approach increases the level of abstraction at which Web Applications have been traditionally developed and integrated.

The main contributions of this paper can be summarized as follows:

- An integration process that, departing from traditional software engineering techniques, extend the views provided by such approaches with a set of new complementary hypermedia views that include server interface definition.
- A set of interaction modes that define the way the user can introduce the values for the set of parameters involved in the service invocation.
- A set of modeling constructs that abstract the definition of One-Step and Multi-Step interfaces.

At this moment efforts are being made towards the support of Compound Services that involve Internet Transactions. OO-H is supported by a CASE tool that, at this stage of development, already provides a model compiler for the automatic generation of interface prototypes. Intensive work is being performed on the OO-H Case tool to provide full support to the method.

References

[1] P. Atzeni and A. Parente. Specification of Web applications with ADM-2. In *Proceedings of First International Workshop on Web-Oriented Software Technology IWWOST'01*, pages 99–129. Valencia University of Technology, 06 2001.

[2] M. Bieber, H. Oinas-Kukkonen, and V. Balasubramanian. Hypertext Functionality. *ACM Computing Surveys*, 31(4), 12 1999.

[3] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. In *Proceedings of the 9th International WWW Conference*, 05 2000.

[4] J. Conallen. Modeling Web Application Architectures with UML. *CACM: Communications of the ACM.*, 42(10):63–70, 10 1999.

[5] O. de Troyer and S. Casteleyn. The Conference Review System with WSDM. In *Proceedings of First International Workshop on Web-Oriented Software Technology IWWOST'01*, pages 30–98. Valencia University of Technology, 06 2001.

[6] Directory Services Markup Language. http://www.dsml.org/.

[7] Inc. Gartner Group. The Future of Web Services: Dynamic Business Webs. Market Analysis, 02 2001.

[8] F. Garzotto, P. Paolini, D. Bolchini, and S. Valenti. Modeling-by-Patterns of Web Applications. In *Proc. International Workshop on the World Wide Web and Conceptual Modeling WWWCM'99*, 1999.

[9] H.W. Gellersen and M. Gaedke. Object-Oriented Web Application Development. *IEEE Internet Computing*, pages 60–68, 01 1999.

[10] J. Gómez, C. Cachero, and O. Pastor. Extending a Conceptual Modelling Approach to Web Application Design. In 12^{th} *International Conference on Advanced Information Systems (CAiSE'00)*, volume 1789, pages 79–93. Springer-Verlag. Lecture Notes in Computer Science, 06 2000.

[11] J. Gómez, C. Cachero, and O. Pastor. Conceptual Modelling of Device-Independent Web Applications. *IEEE Multimedia Special Issue on Web Engineering*, 8(2):20–32, 04 2001.

[12] N. Koch and L. Mardiel. State of the Art and Classification of Electronic Product Catalogues on CD-ROM. *Electronic Markets*, 7(3):16–21, 1997.

[13] F. Manola. Technologies for a Web Object Model. *IEEE Internet Computing*, pages 38–47, 01 1999.

[14] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of XML. *IEEE Data Engineering Bulletin*, 42(10):63–70, 10 1999.

[15] S. Murugesan, Y. Deshpande, S. Hansen, and A. Ginige. Web Engineering: A New Discipline for Development of Web-based Systems. In *Proceedings of First ICSE Workshop on Web Engineering*, 05 1999.

[16] D. Schwabe and R. Almeida Pontes. A Method-based Web Application Development Environment. In *Proceedings of the 8th International WWW Conference*, 1999.

[17] Simple Object Access Protocol. http://www.develop.com/soap/.

[18] Universal Description, Discovery and Integration. http://uddi.microsoft.com/.

[19] User Interface Modeling Language. 16. UIML http://www.uiml.org.

[20] UML Specification. V1.3. http://www.rational.com/uml/index.jsp, 06 1999.

[21] OMG Unified Modelling Language Specification. http://www.rational.com/uml/, 06 1999.

[22] Web Services Description Language. http://msdn.microsoft.com/xml/general/wsdl.asp.

[23] WSUI Specification. Working Draft. V1.0. http://www.wsui.org, 10 2001.