

# Assertions to Better Specify the Amazon Bug

L. Baresi, G. Denaro, L. Mainetti, and P. Paolini  
Dipartimento di Elettronica e Informazione  
Politecnico di Milano  
Piazza L. da Vinci 32, I-20133 Milano, Italy  
baresil|denaro|mainetti|paolini@elet.polimi.it

## ABSTRACT

Modern Web applications are mainly distributed systems that exploit the Internet as communication means and the Web as neutral interface to access services and data. The addition of services to Web applications poses problems that are usually tackled at the technology level, but that should be addressed during design to deliver quality Web applications. A typical example of these problems is the Amazon bug, an annoying problem that the user could encounter if after adding products to his shopping cart, he rolls back to a page with a previous version of the cart and tries to buy it. This would make the user buy the last version of the cart's contents, which in some subtle cases could be different from what expected.

In this paper, we do not want to discuss all design aspects, but only how provided services/operations should jointly be designed with the rest of the system. We propose a new reference model for Web applications: Operations require a more complex model where they are not simply appended to information and navigation elements, but they can cooperate with them. Besides the reference model, the paper proposes the use of assertions to constraint the behavior of designed operations. Assertions do not only predicate on how data should be modified, but must also take into account how presentation and navigation could be affected by the execution of the operation.

## Keywords

Web application design, Web operation, assertions

## 1. INTRODUCTION

Modern *Web applications* are mainly distributed systems that exploit the Internet as communication means and the Web as neutral interface to access services and data. These applications, in many cases, change the perspective of what the Web is: We are moving from applications where the Web was the key feature (e.g., data-intensive systems) to applica-

tions for which the Web is a means to supply services ([3]).

This new perspective is appealing for several different reasons. The Internet offers a highly available neutral (with respect to application domains and implementation technology) communication means. The Web is easy to access and does not require special-purpose client-side technology: A simple browser is enough to connect to the server and exploit its services. Moreover, off-the-shelf technology (middleware, markup languages, and enterprise components) can be used to plug new services and make the system scale according to the actual needs ([14, 12]). The Web offers also a fancy way for implementing more engaging user interaction. Essentially, operation-oriented interfaces, which offer the minimum interactivity, are replaced by more user-oriented interfaces to let the user mix navigation, data access, and service invocation in a natural way.

The Web could simply introduce a visual make-up of the application through a Web interface, without improving the ergonomics and interaction paradigm. This applies to the attempts of porting legacy systems on dedicated intranets (e.g., accounting and billing systems). The Web can also imply a stronger coupling: The application becomes an homogeneous mix of navigation and services. This applies to new applications, like the e-services available on the Internet, or to completely rethought systems. All important aspects must be revised to fully exploit the Web in engaging users and improving accessibility and usability.

In both cases, the addition of services to Web applications does not come for free ([17, 19]). It poses problems that are usually tackled at the technology level, but that should be addressed during design as well. Our strong belief is that quality Web applications come from good design: Improving our design standards and our ability for validating the design are today crucial factors to improve the quality of Web applications. A typical example of these problems is the *Amazon bug*, an annoying problem that the user could encounter if after adding products to his shopping cart, he rolls back to a page that displays a previous version of the cart and tries to buy it. This would make the user buy the last version of the cart's contents, which in some subtle cases could be different from what expected. A proof that even famous Web applications are prone to errors and could be improved if carefully designed.

In this paper, we do not want to discuss all design aspects,

but only how provided services/operations should jointly be designed with the rest of the system. The first consequence is the need for a new reference model for Web applications: The conventional layered view typical of data-intensive Web sites is not enough anymore. Operations require a more complex model where they are not simply appended to information and navigation elements, but they can cooperate with them. Thus, the paper sketches a new *reference model*, mainly inspired by HDM/W2000 (Hypermedia Design Model,[7, 1]), but whose main concepts could be applied to all other similar approaches.

The proposed model allows designers to think of navigation and services jointly, trying to foresee how they can influence each other to avoid mutual side effects that could jeopardize the system. Besides the reference model, the paper proposes the use of *assertions* to constraint the behavior of designed operations: pre- and post-conditions and invariants are the standard way for specifying the behavior of operations. Roughly, pre- and post-conditions define what must hold true in the system before and after executing the operation; invariants define properties of application elements that must always be true. The use of natural language is the simplest way to express them, but more specific notations, like *iContract* [13], *Eiffel* [16], and *OCL* [8]<sup>1</sup>, would guarantee a higher degree of rigor. To the best of authors' knowledge there is neither a proposal that is specific to hypermedia systems, nor an approach for blending hypermedia and operations. Assertions can not only predicate on how data should be modified, but must also take into account how presentation and navigation could be affected by the execution of the operation.

This is why, the paper identifies the main requirements for such an assertion system. It does not aim at proposing a completely new approach, but rather it is based on *OCL*, the standard UML assertion language. The additions are partially motivated by and exemplified on the Amazon bug.

Finally, the presentation follows a user-oriented perspective. We want to clearly state what the application should do with respect to its users. The system-oriented perspective could introduce further requirements and constraints that are hidden to users. The two design aspects are obviously correlated, in the sense that latter must consistently implement the former, but are also different from a methodological perspective.

The rest of the paper is organized as follows. Section 2 motivates the paper by introducing the Amazon bug, a typical annoying behavior that we found in the well-known e-commerce site and in several other applications. The purpose is not to blame Amazon but to convince readers. Section 3 paves the ground to assertions by proposing a first taxonomy of Web operations, identifying the main new requirements as to design models, and sketching the new reference model. Section 4 describes how assertions can be exploited to define the behavior of Web operations and explains how

<sup>1</sup>*iContract* is an assertion language for the Java programming language. *Eiffel* is an object-oriented language that supplies assertions as built-in concepts. *OCL* is part of the UML family and associates model objects with constraints at design time.

they could solve – how their use would have avoided – the Amazon bug. Finally, Section 5 concludes the paper and presents our future work.

## 2. THE AMAZON BUG

The annoying behavior discussed in this section is only an example, taken from the well-known e-commerce site, of a class of (subtle) problems that unfortunately belong to many Web applications. We exemplify the problem using the Amazon site only because of the site's renown - we neither want to blame Amazon, nor we want to complain. Our intention is rather to warn readers that similar bugs can be everywhere.

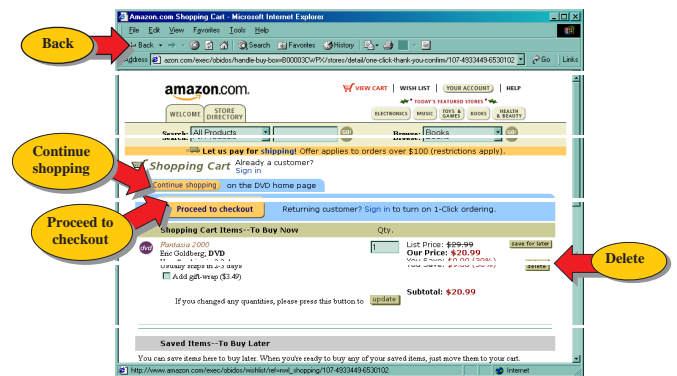


Figure 1: The shopping cart with one DVD

The example is simple, but sufficient to understand what types of situations users may stumble on. The user wishes to buy some DVDs and browses through the catalogue to select what he wants to buy. After choosing the first DVD, the application presents to the user the shopping cart with the selected item (say, the *Fantasia 2000* DVD) (Figure 1). From his shopping cart page (Figure 1), the user can:

- Check its purchase out, using the proceed to checkout button;
- Return to shopping, using the continue shopping button;
- Cancel selected items, using the delete button;
- Use the browser to navigate back, using the back button.

In the latter case, the user would not exploit the application logic, instead he would benefit from application-independent capabilities offered by the browser.

If after *Fantasia 2000*, the user wanted to buy also *Fantasia*, he could use continue shopping, locate the DVD in the page that lists all Disney DVDs (step 1 of Figure 2), and check it out for purchase. The selection makes the user move to the shopping cart page (step 2 of Figure 2), which now contains two DVDs: *Fantasia* and *Fantasia 2000*.

Thinking of price, the user changes his mind and decides that he does not want to buy the second DVD, i.e., *Fantasia*,

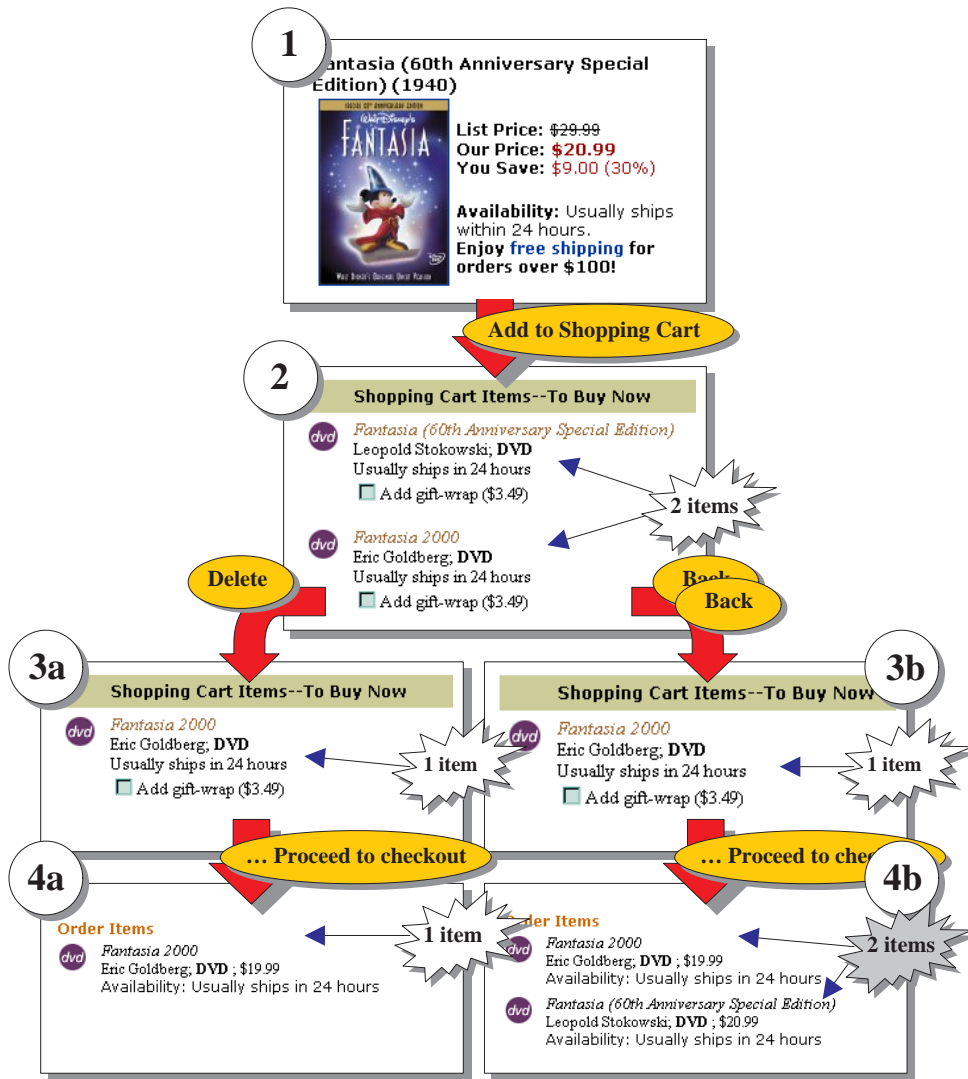


Figure 2: Different alternatives to complete the purchase

any more. The rigorous user (that is, the user who follows the suggested application logic) would delete the DVD from the shopping cart, using the delete button (step 3a of Figure 2) and then he would checkout the shopping cart with just *Fantasia 2000* (step 4a of Figure 2).

In contrast, the “normal” user (the typical “free” navigator) would roll back the session to the *previous* shopping cart, that is, the one with one item, using the back button of his browser twice: The first time to return to the page listing all Disney DVDs; the second time to bring the user to the previous cart, that is, the page shown as step 3b of Figure 2, which contains only *Fantasia 2000*. Thus, he would check out this page/cart with the surprise that he is about to buy the two DVDs (step 4b of Figure 2), instead of just *Fantasia 2000*.

The anomaly is twofold. By clicking the proceed to checkout button in the page that contains one item only (step 3b), the user ends up buying two items (step 4b), unless he stops the process. By clicking the same button - proceed to checkout - in two visually identical pages (steps 3a and 3b of Figure 2) the user gets two different operational results: He buys one item in the first case, and two items in the second case.

### 3. COPING WITH WEB OPERATIONS

The purpose of this section is to discuss the relevant issues for designing Web-based operation/services. In this paper, we do not distinguish between operation and service: We consider them as synonyms, even if we could say that a service comprises several operations.

The Amazon bug demonstrates that the coexistence of different interaction paradigms, i.e., navigational and operational, introduce mutual side effects and may lead to anomalous behaviors and unpredictable effects for the users.

Operations may take control over standard navigation, taking the user to different pages within the hypermedia navigation schema. Usually, operations can be invoked from a page, but lead the user through different pages during their execution.

The sequence of information items (nodes) used by operations should be based on the intrinsic navigation possibilities of the application, both for usability and accessibility reasons and to support reuse. In other words, if *item2* follows *item1* in a given operational context, the user naturally expects to find also a navigational connection between the two items, and the designer should be able to exploit and reuse this underlying navigational structure.

Free hypermedia navigation, directly activated by the user while browsing the interlinked information structures, may disrupt the standard flow of information required by the execution of an operation. Think of, for example, the *back* and *forward* buttons supplied by the browser.

These examples clearly indicate the need for a design approach that help designers master this complexity and reduce the risk of introducing inconsistencies and unpredictable behaviors. Unfortunately all known design approaches are suitable to either data-intensive Web applications, that is,

are hypertext oriented, but do not allow for modeling real operations (for example, [7, 10, 6, 9, 20]), or come from “traditional” software engineering and do not pay enough attention to hypermedia. Just a couple of approaches ([2, 18]) address operations, but are limited to “simple” input/output operations; only Conallen [4] tries to intertwine the two domains, but with a very implementation-oriented viewpoint.

#### 3.1 A preliminary taxonomy of Web operations

Before introducing our new reference model, let us try to identify a preliminary taxonomy for Web operations and point out the requirements that they induce. Web operations can allow users to:

**Select the actual parameters to complete an operation.** In many cases, while the user browses the information repository, he also collects those data that will become the actual parameters of his operation. For example, when we select a book we want to buy, we identify it as the parameter of the *buy* operation. Even if the user perceives he is navigating, he changes the way available information is organized (according to the HDM jargon, he is changing the collections defined for the application). This may lead also to changing the state of the pointed element (e.g., the application immediately decrements the number of available copies of the selected book) and the way users can navigate through the application (e.g., the application forbids users from navigating through the pages of non-available books).

**Change the way users can navigate through pages (data).** Even if operations do not change the application contents, they can guide the user while navigating through the pages. Specific choices could allow some links, but disallow others. For example, the operation could change the order used to present the elements of a collection: Books could be presented by title or by author's name. In this case, we would not change the elements in the collection, but simply the links among them.

**Enter new data in the system.** For example, all pages that embed forms implicitly provide these operations. But this means that if we have forms, the application data may be augmented and changed. It could be the case also that not all inserted information becomes “navigable”, i.e., it is not rendered in Web pages. In many cases, when we supply a system with our personal data to register ourselves, we cannot browse them.

**Perform complex state-aware computations.** For example, consider those applications that log their users and adjust what they can do with respect to what they have already done. Otherwise, we can mention those applications that have a high degree of computations, like billing or special-purpose applications. These operations must exploit *computational* “objects”: They can simply be used to store the state of the computation or of the particular user, but they can also be delegated to compute some complex algorithmic tasks that could hardly be striven on a DBMS.

Besides these classes, we can also think of other operations that we do not want to consider now. The first group can be seen as fake navigation: for example, when users graph-

ically reshape the elements they are browsing. The second group comprises all those operations that we could term as *advanced*, that is, those that deal with customizing the application with respect to specific contexts ([11]): devices, user profiles, quality of service, etc. Orthogonally, we should model the fact that sets of operations should be considered *logical transactions*: This is an important aspect, but it is not covered in this paper. Interested readers can refer to [5] for a complete presentation.

### 3.2 A new reference model

This partial taxonomy poses a set of clearly identifiable requirements to the supporting design model, which must encompass all features that are common to hypermedia systems, but, at the same time, must integrate operations and the effects they have on the whole system. Figure 3 presents a simplified – to concentrate on the main concepts – *reference model*<sup>2</sup> for *Web applications*: It borrows the layers from data-intensive systems, namely HDM/W2000, and adds new elements to cope with the problems introduced so far. For the sake of clarity, new elements are depicted with a gray background.

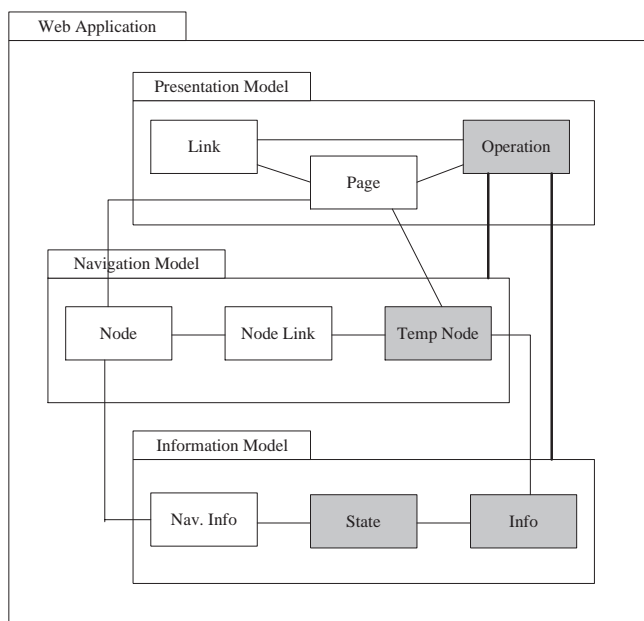


Figure 3: Our reference model for *Web applications*

In this paper, we refer to HDM as underlying starting point, but the layered organization of Figure 3 with *Information*, *Navigation*, and *Presentation* models applies to almost all known approaches. These layers can be seen as an application to the hypermedia domain of the well-known *model-view-control* design pattern.

Operations can change the application data, but also – or only – the way they can be navigated or presented. The box *operation* has been added in the *Presentation* model, because it is the level at which the user perceives operations, but they can work at all levels. Its associations state that

<sup>2</sup>According to the UML jargon, Figure 3 describes the meta-model of Web applications.

an operation can work on/with *pages* and *links*, but also with all the elements in the other two models<sup>3</sup>.

The equation that modeled information must be browsable is not true anymore. Operations, and more generally Web applications, need to model both navigable information, but also the data required by supported computations. To cope with this requirement, we distinguished between *nav info* and *info* to explicitly consider the two different uses of the modeled information.

Not all data must be persistent anymore. Temporary data, and thus nodes and pages, could be required to build permanent information by following a given process. To cope with this problem we added *temp* nodes that can be used to model temporary information items and then be rendered into special-purpose pages.

Application data cannot be state-less anymore. Theoretically, each information element could be associated with a set of states through which it can evolve during its life. States could also become important to implicitly define which elements belong to which collections. This is why we added *states* as meaningful properties of modeled information. But now, since data can change, pages do not embody information elements, but they simply render them. This means that the assumption that a page *is* an information element is not correct anymore, but a page is simply a materialization of an information element in a precise instant.

## 4. ASSERTIONS AND THE WEB

Assertions, as proposed by Bertrand Meyer [16], can be seen as *contracts* between the service provider and the customer, that is, the “entity” that will use the service. These contracts are a light and precise way for specifying what must hold true in the system. Assertions can be expressed using natural language, but more precise notations increase the quality of delivered design.

The following sections explain how we propose to extend assertions and how they can be used to better specify the Amazon bug at design level.

### 4.1 Assertions and Web Operations

This section presents the basic extension to OCL specific to Web applications. Since operations are dynamic entities, the actual elements related to an operation are expected to vary while the operation executes. For example, in many cases the current page at the beginning of the operation differs from the current page at the end of the operation itself. Moreover, the integrity relationships of the elements distributed among the different application layers cannot be guaranteed a-priori. On the contrary, it is part of the operation semantics to state whether the coherence among elements in different application layers must be ensured or, if not, which of the available representations must be used. To model the possible cases, operations are associated with a set of formal parameters ([15]). This enables designers to constrain the desired behavior by stating whether the actual

<sup>3</sup>Instead of transforming the model in a kind of “spaghetti” diagram, we used thick lines to define an association between an *operation* and all elements in a model.

parameters of an operation come from the current page, or the (current) internal state, or the two representations must be consistent.

Operations can be of two types: *low level operations*, i.e., single activities, and *high level services* associated with sequences of both user actions and low level operations. Low level operations are atomic and they can be specified in terms of pre- and post-conditions. High level operations are not atomic: Invariants can be used to describe properties that are expected to be true at each point of the execution of the service.

The *reference model* as to operations is presented in Figure 4 and can be considered a kind of zoom with respect to the model of Figure 3. This model is hidden to designers, but it is needed to define the semantics of the OCL extensions, i.e., we explain each extension by means of plain OCL constraints on the reference model.

The OCL extensions relevant to the Amazon bug are presented below. The new keywords are prefixed with *wcl* (that is, *WebOCL*).

**wclCurrentPage() : Page** In the context of<sup>4</sup> a Web operation *op*, this function returns the page currently visualized. The precise semantics is:

$$wclCurrentPage \triangleq op.currentpage$$

**wclIsVisualized(i: Item)<sup>5</sup> : Boolean** In the context of a Web operation *op*, this function results true if the element *i* is visualized in the current page. The precise semantics is<sup>6</sup>:

$$wclIsVisualized(item) \iff op.curPage.section.sctElement -> exists(item)$$

Note that in HDM *sections* are independent parts of pages and *section elements* are the items that each section contains (See [1] for a more complete presentation of the HDM/W2000 framework).

**wclIsStored(i: Item) : Boolean** In the context of a Web operation *op*, this predicate results true if the element *i* is currently stored in the application state. The precise semantics is:

$$wclIsStored(item) \iff op.ciState.navElement -> includes(item) \text{ or } op.ciState.infoElement -> includes(item)$$

**wclActualParameterLikePresented(i: Item) : Boolean** In the context of a Web operation *op*, this predicate results true if the actual value of *i* comes from the current application state. The precise semantics is:

$$wclActualParameterLikePresented(item) \iff op.formalPar -> includes(item) \text{ and } wclIsVisualized(item)$$

<sup>4</sup>This means that the *function* is available within the assertions of an operation.

<sup>6</sup>Due to space limitations, some references to Figure 4 have been abbreviated: *current page* with *curPage*, *current initial state* with *ciState*, and *formal parameter* with *formalPar*.

**wclActualParameterLikeStored(i: Item) : Boolean** In the context of a Web operation *op*, this predicate results true if the actual value of *i* comes from the current page. The precise semantics is:

$$wclActualParameterLikeStored(item) \iff op.formalPar -> includes(item) \text{ and } wclIsStored(item)$$

**wclActualParameterUpToDate(i: Item) : Boolean** In the context of a Web operation *op*, this predicate results true if the actual value of *i* is both in the one from current page and the one from the current application state. The precise semantics is:

$$wclActualParameterUpToDate(item) \iff wclActualParLikeStored(item) \text{ and } wclActualParLikePresented(item)$$

**wclLinkFollowed(l: string) : Boolean** In the context of a Web operation *op*, this predicate results true if the link *l* is followed. The definition of a precise semantics for this predicate would need a proper event model for Web applications. Here, we assume that for all links at *Presentation* layer, hereafter *presentationLink*, a corresponding event is defined. We also assume that it is possible to know whether or not an event is fired by means of the primitive predicate *followLinkIsFired*. The definition of a proper event model can be found in [5]. The semantics of *wclLinkFollowed* is:

$$wclLinkFollowed(stringlinkname) \iff exists(l : PresentationLink | l.name = linkname \text{ and } l.wclIsVisualized() \text{ and } l.followLinkIsFired())$$

## 4.2 How assertions specify the Amazon bug

Given the extensions defined in the previous section, the Amazon bug can be solved in several different ways. Actually, it is not a real solution, which depends on the technology used to implement the application, but it is a precise way to start thinking of that since the early design phases.

The model fragment of Figure 5 introduces the mandatory elements to explain the Amazon bug. A *CartPage* page type defines how we want to visualize a cart. This page type renders the *ShoppingCart* collection type, which is the way we model the cart at the information/navigation level. The cart is a set of DVD objects (entities according to the HDM jargon). The *Center* stores the quantity in the cart of each bought item. Each cart belongs to a *Customer*.

The Amazon bug comes from an assumption that was correct for state-less applications: *A page is always a correct visual materialization of the corresponding node (information element)*. Nowadays, Web applications can modify their data and thus a page can become *old* with respect to the information contents it materializes. This constraint could be modeled as an invariant associated with the *CartPage* page definition:

$$context CartPage \\ inv : wclIsStored(self.renderers)$$

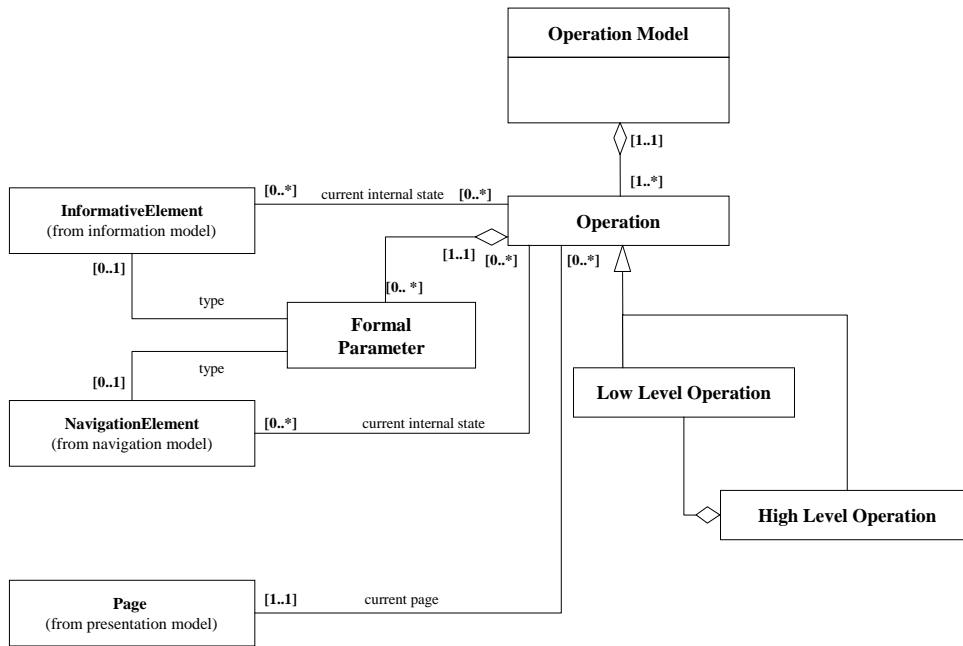
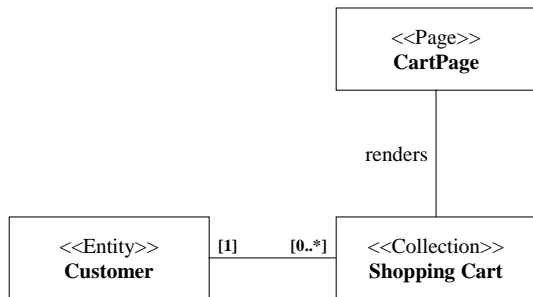


Figure 4: Reference model for operations

Where, `wclIsStored` is the predefined function that states the consistency between the body of the page and the state of the node it renders. Since we used an invariant, we would state that the page must always be consistent: it will be responsibility of implementations to guarantee that pages stacked in the browser's history are correct when accessed. A less stringent requirement could be that the pages must not be stacked by the browser. This would require a dedicated invariant to represent that the page once visualized must not be added to the navigation history.

A further possibility is the definition of the constraint as a property of the operation that could use the page. Instead of predicating on a property that must always hold true, we limit our attention to a particular operation. If we consider the buy operation, we could say:



```

contextbuy(s : ShoppingCart, c : Customer)
pre :
    wclActualParameterUpToDate(s) and
    wclLinkFollowed("Proceedtocheckout")
post :
    wclCurrentPage = SupplyUserDataPage
  
```

Figure 5: An excerpt of the model of an Amazon-like application

Finally, we could change the perspective and consider that the actual visualized information is always correct. In this case, for the buy operation, we could say:

```

contextbuy(s : ShoppingCart, c : Customer)
pre :
    wclIsVisualized(s) and
    wclLinkFollowed("Proceedtocheckout") and
  
```

```

wclAcualParameterLikePresented(s)
post :
wclCurrentPage = SupplyUserDataPageand
wclIsStored(s)

```

To complete these assertions, we should also say that the products in the shopping cart *s*, e.g., DVDs, must be available and the customer *c* must be registered. These further constraints would only make the pre- and post-condition more complex, but they do not change the approach.

## 5. CONCLUSIONS AND FUTURE WORK

In this paper, assertions are used to better specify the Amazon bug at design level, but our strong belief is that their applicability is more general and can address a wider class of problems. The main thesis of this paper is that the careful design of Web operations is the best way to avoid annoying behaviors and increase the quality of released applications. We briefly describe the main requirements that arise from the integration between the Web and “real” operations. We sketch a new reference model for accomplishing these requirements and reason on *Web applications*. We propose also how to exploit assertions to specify Web operations and we exemplify our approach on specifying the Amazon bug. Needless to say, the use of assertions is not new in software engineering, but to the best of authors’ knowledge, this is the first time they have been applied in the context of Web applications. Our proposal is part of a multidimensional design approach that clearly distinguishes between user- and system-oriented design and orthogonally considers information, navigation, and presentation design in a coordinated way.

Our future work will consist in completing and refining our proposal to extend assertions (OCL) to specify Web operations. We will complete the set of primitives presented in this paper, but we will investigate also other problems, like customization and user profiles, that are not considered now.

## 6. REFERENCES

- [1] L. Baresi, F. Garzotto, and P. Paolini. From web sites to web applications: New issues for conceptual modeling. In *Proceedings of the International Workshop on The World Wide Web and Conceptual Modeling, co-located with the 19th International Conference on Conceptual Modeling*, 2000.
- [2] A. Bongio, S. Ceri, P. Fraternali, and A. Maurino. Modeling data entry and operations in webml. In *WebDB (Informal proceedings) 2000*, pages 87–92, 2000.
- [3] G. Booch. The architecture of web applications, 2001. [http://www.developer.ibm.com/library/articles/booch\\_web.html](http://www.developer.ibm.com/library/articles/booch_web.html).
- [4] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, Reading, 1 edition, 1999.
- [5] G. Denaro. Extending OCL to meet web application requirements. Technical report, Politecnico di Milano, 2001.
- [6] P. Fraternali and P. Paolini. Model-driven development of Web applications: the AutoWeb system. *ACM Transactions on Information Systems*, 18(4):323–382, 2000.
- [7] F. Garzotto, P. Paolini, and D. Schwabe. HDM – A model for the design of hypertext applications. In *Proc. of ACM Hypertext’91, Hypertext – Integrative Issues*, page 313, 1991.
- [8] O. M. Group. Object constraints language specification, Feb. 2001.
- [9] R. Hennicker and N. Koch. A UML-based methodology for hypermedia design. In *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939 of *LNCS*, pages 410–424. Springer, 2000.
- [10] T. Isakowitz, E. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, Aug. 1995.
- [11] G. Kappel, B. Proll, W. Retschitzegger, W. Schwinger, and T. Hofer. Modeling ubiquitous web applications - a comparison of approaches. In *Proceedings of the Third International Conference on Information Integration and Web-based Applications and Services (iiWAS2001), Linz, Austria*, pages 163–174, sep 2001.
- [12] N. Kassem and The Enterprise Team. *Designing Enterprise applications with the Java 2 Platform*. Addison-Wesley, 2000.
- [13] R. Kramer. iContract – The Java Design by Contract Tool. In *Proceedings of TOOLS26: Technology of Object-Oriented Languages and Systems*, pages 295–307. IEEE Computer Society, 1998.
- [14] F. Manola. Technologies for a web object model. *IEEE Internet Computing*, jan 1999.
- [15] A. M. Memon, M. E. Pollack, and M. L. Soffa. Automated test oracles for GUIs. In *Proceedings of the ACM SIGSOFT 8th International Symposium on the Foundations of Software Engineering (FSE-00)*, volume 25, 6 of *ACM Software Engineering Notes*, pages 30–39. ACM Press, Nov. 2000.
- [16] B. Meyer. *Eiffel: The Language*. Object-Oriented Series. Prentice Hall, New York, N.Y., 1992.
- [17] S. Murugesan and Y. Deshpande. ICSE’99 workshop on web engineering. In *Proceedings of the 1999 International Conference on Software Engineering*. IEEE Computer Society Press / ACM Press, 1999.
- [18] O. Pastor, V. Pelechano, E. Insfran, and J. Gomez. From object oriented conceptual modeling to automated programming in Java. *Lecture Notes in Computer Science*, 1507, 1998.
- [19] T. A. Powell. *Web Site Engineering: Beyond Web Page Design*. Prentice Hall, Upper Saddle River, New Jersey, 1998.
- [20] D. Schwabe and G. Rossi. An object oriented approach to web-based applications design. *Theory and Practice of Object Systems*, 4(4):207–225, 1998.