Journal of Web Engineering, Vol. 6, No.1 (2007) 019-048
© Rinton Press

ENGINEERING WEB APPLICATIONS USING ROLES

GUSTAVO ROSSI

Facultad de Informática, Universidad Nacional de La Plata and Conicet Argentina
gustavo@sol.info.unlp.edu.ar

JOCELYNE NANARD, MARC NANARD

LIRMM, CNRS/Univ. Montpellier, France
nanard@lirmm.fr

NORA KOCH
Institut für Informatik, Ludwig-Maximilians-Universität München and

FAST GmbH, Germany
nora.koch@ifi.lmu.de, koch@fast.de

Received February 22, 2005
Revised August 10, 2006

Although role modeling is a topic that has been treated over years in the object-oriented community, its use in the life cycle of Web Engineering, and particularly in object-oriented Web design methods, has been seldom discussed and used yet. In this paper, we introduce roles in the modeling and design armory of existing Web engineering methods and show how it improves their expressive power and help to solve design problems that appear frequently in Web applications. We first survey the state of the art of Web engineering modeling approaches. A simple example is used to point out some situations in classic Web engineering modeling where it is not possible to express that objects or nodes should change their properties (attributes or behaviors) according to the collaborating subject (the objects which send them messages or the nodes which are linked to them). Next, we introduce the object-oriented role concept and discuss how it has been used so far in the software engineering community and how it can be useful for Web engineering modeling. Existing methods (like UWE and OOHDM) are used as an example to show how to introduce roles during the Web engineering process. We compare our approach with others and conclude with some further research we are pursuing.

Key words: Web application, engineering, modeling, role, collaborating objects
Communicated by: D Schwabe & O Pastor

1 Introduction

The increasing use of the Web as a software platform together with the advance of technology has given raise to a completely new generation of Web Applications; these applications allow ubiquitous access from fixed and mobile devices, provide personalized features to individuals, support complex business processes and workflows, etc. The Web engineering community has already discussed and hopefully solved many of the design problems arising from this complexity [7, 11, 27, 42]. In this paper we focus on one important issue that has been so far ignored (or at most only partially addressed): how to specify, using a high level notation, those object or node aspects which vary according to their involvement in different collaboration or link relationships. Particularly, we are interested in how to model object and node classes in which there are properties (attributes, behaviors, outgoing links) which are intrinsic to the class and others which depend on the collaborating object (i.e., the object which sends a message) or the incoming link.

Usually, application objects must be designed to provide different services (data, behavior) according to the context in which they are accessed. For example, suppose an academic application used by the administration staff; when a person gets enrolled in the university, she is treated as a student. This means that the corresponding object has the message protocol of students. Later, the same person may then become a teaching assistant in a course or an employee of the university: the corresponding software object(s) will presumably exhibit slightly different behaviors. The same person might, at the same time, behave as a student AND as a teaching assistant. Accordingly, the corresponding software object must behave both as a teacher and as a student as if it belongs to two different classes.

In a typical e-commerce Web application, the same product object will provide different services according to the application object interacting with it, e.g., in the context of a stock application, when being part of an order, when accessed as a recommended item, when treated as a gift to an employee, etc. In the www.amazon.com Web site, for example, when we access a CD in our recommendation list, the CD exhibits a link that explains why the CD is recommended. However, if we access the same CD following another path (for example from the list of novelties), the link is not available.

Notice that, in these two examples, application objects (persons, products) or nodes (CDs) vary their features when interacting, collaborating, or being accessed by other objects or nodes, as if they were dynamically changing their base class. This variation occurs not only when we send specific messages to those objects, but also when we navigate them, even in simple Web applications. The example above shows only a small sub-set of the different kinds of situations in which some *semantic flexibility* is needed for modeling.

As we show later in the paper, we can not solve the problem by using a naïve combination of object-oriented abstraction primitives (such as building a class hierarchy of Persons, Employees, Students, etc.) because class hierarchies do not support objects (or nodes) belonging to two different classes or changing their class dynamically.

Some of the situations above could eventually be solved (at the design level) by using combinations of well-known object-oriented primitives or patterns (See for example [41]). However, it is important to deal with these concerns at the correct level of abstraction; in particular, we need a clear way to express this kind of variability at a higher level, while modeling the application. We argue that we can solve some of the modeling problems, in an elegant and clear way, by introducing the role concept.

Informally, a role is the set of properties which are important for an object to behave in a particular context (or when participating in a certain relationship). When the object behaves as expected in that context, we say that the object is playing a role. In everyday life, we usually say that a person is playing a role (e.g., student) when he acts as expected when being a student. We know by our experience that the same person might play different roles according to whom he is interacting (e.g., the student might play the role of son, when interacting with his parents). However, there are certain, say “intrinsic”, properties of the person (his name, age, etc.) that are independent of the role, while others only make sense when playing the role (e.g., the student number).

In software meanwhile, a CD in a store’s catalogue may play different roles according to who (which other object) is interacting with it. While its intrinsic properties (e.g., name, performer, etc.) will be shared in every role, it might exhibit different properties and behaviors when being accessed by different objects.

Surprisingly, the role concept has been rarely used so far in the hypermedia community and thus has been barely ignored in the Web Engineering community. Remarkable exceptions are the work in [1] and [11], which, though a bit different from our approach, will be addressed in the Related Work section.

We argue that the role concept is critical in the context of Web applications because:

- Web applications embody complex and comprehensive domain models (which are then used by different specific applications), and thus it is usual that the same application object might have to assume different roles during the application life cycle or in the context of different applications.
- Being the Web a realization of the hypermedia paradigm, a model in which relationships are fundamental, roles represent an important tool for expressing node’s variations when involved in different relationships.

In this paper, we discuss why roles should be used as first-class citizens in Web design methods, and propose some simple ways of introducing them in some well-known design approaches, like UWE [27] and OOHDM [43]. The contributions of this paper are two fold:

- We show that the introduction of the role concept improves existing methods. In doing so, we aim to make the Web Engineering community aware of some modeling problems and we explain how to deal with different design concerns using roles.
- While comparing our approach with others using roles, we clearly show when roles should be used and when other design structures or patterns are preferred.

The rest of the paper is organized as follows. We first discuss the state of the art of Web Engineering modeling approaches and show some drawbacks of pure “class-based” approaches. Next, we introduce the concept of roles and briefly compare our notion with others in the literature. Then, using a common example, we show how to introduce roles in existing Web applications design methods. Finally we compare our work with other similar approaches and present some further research we are pursuing.

2 Modeling Web Applications. State of the Art, Approaches, and Problems

In this section, we explain how existing methods model and design a Web application; our intent is not to discuss minor syntactic differences between design methods, but to show which are the common primitives and abstraction mechanisms and identify a set of design problems that remain unsolved. Throughout this paper, we use a simple example to show why the concept of roles is needed (in different development stages) and how we use it in Web engineering methods. Suppose a Museum of modern art which organizes exhibitions of artworks and at the same times acts as an Auction house for those artworks whose owners are interested in selling their artworks. From the application point of view, we must support the organization of exhibitions, i.e., indicating the room in which each artwork will be placed, and the organization of auctions. Auctions can take place in different places and a calendar of auctions for a particular artwork should be maintained. Artworks may be in restoration; in this case, besides the basic artwork information, we aim to know the restorer's identity and the date in which the restoration will finish.

We aim to build different Web applications according to the intended task. In an application for exploring the museum virtually, users can navigate all artworks (even those who were sold), information about artists, restorers, etc. They can even add comments on artworks that might be useful for administrator to assess which artworks are more "popular". Another Web application must support the work of administrators, allowing to "tag" artworks to be sold in auctions, included in exhibitions, or put into restoration. Expert evaluations are also dealt with in this application. Finally, we might be interested in an application to proceed with the auction on-line (as in www. sothebys.com). For the sake of clarity, we will next present the most important modeling and design activities in existing Web Engineering methods and then argue why more design primitives are needed by analyzing some problems which we illustrate on the Web museum example. A summary of the design problems is presented in a separate sub-section.

2.1 Design Activities in Web Design Methods and Analysis of the Problem

Web application development methods like UWE [27], OOHDM [43], OO-H [16], OOWS [35], WebML [7], and WSDM [11] partition the development space in (at least) five different activities: requirement gathering and specification, conceptual, navigational, and presentational design, and implementation. They usually provide modeling primitives for at least the first three design activities. Some of these methods (UWE, OOHDM, OO-H and OOWS) are based on the object-oriented paradigm, i.e., they integrate structure and behavior in the same component (a class). Others, like WebML and WSDM, are based on well-known data modeling approaches, like the E/R [8], or the ORM [22]. The conclusions of this paper can be easily applied to any object-oriented approach, and therefore, to focus our discussion we assume that the basic modeling primitives are those in the object-oriented paradigm. A discussion on the application of these ideas to other, non-object-oriented, approaches, though interesting, is outside the scope of this paper. For the sake of conciseness, we focus on conceptual and navigational design and ignore subtle notation differences between existing approaches.

2.1.1 Conceptual Design

There is a consensus in Web Engineering that, during conceptual design, a model of the underlying application domain is built; application objects, behaviors, relationships, and eventually business processes are specified using well-known notations such as UML [34] or the E/R model. Some approaches consider the conceptual model as "just" a content or data model [7]. Others [27, 35, 43] treat the conceptual model as a full-fledged domain model which must include in a cohesive way all intended object behaviors, their relationships with application objects, together with their dynamics and state changes. In this paper, we will assume the latter interpretation. In general, the conceptual model is built as being navigation-neutral, i.e., navigation design issues are not included in the conceptual model. For data modeling approaches, this decision is natural as the conceptual model only specifies contents. Object-oriented approaches tend to specify the conceptual model as a general domain model which can be used regardless the navigation topology specified in later stages, which usually implies analyzing the core domain behaviors and properties.

A simplified class model for our example application is shown in Figure 1. For conciseness, we omit the presentation of sequence diagrams, state models, and other artifacts needed to completely specify the domain.

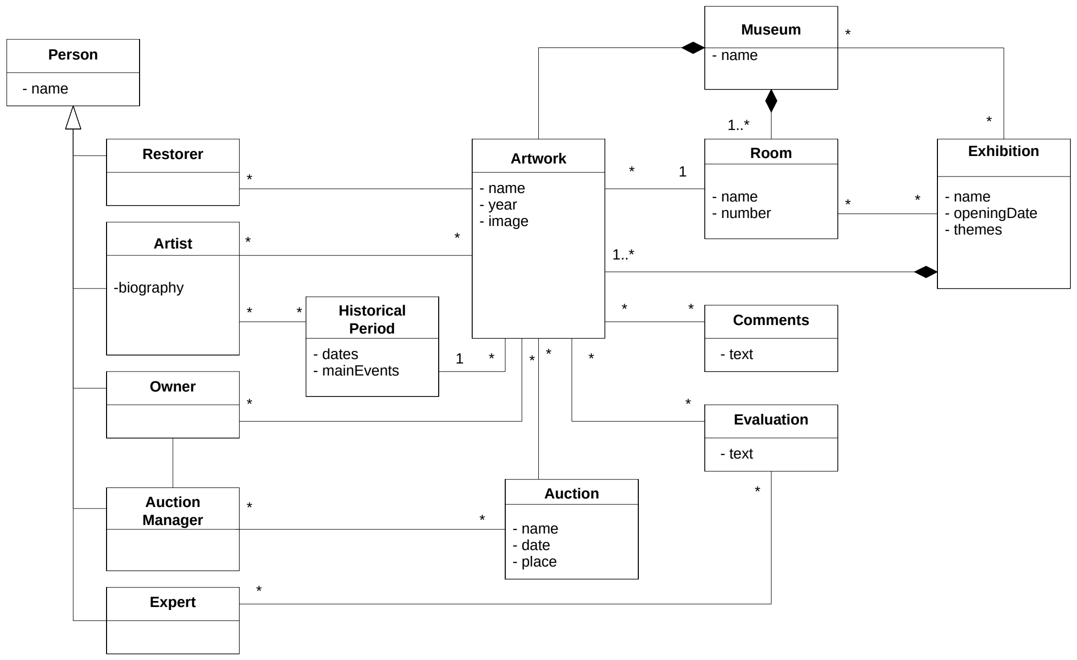


Figure 1 Conceptual model of the museum application.

The conceptual model in Figure 1, as it may result from a "conventional" object-oriented method, shows that the Museum offers *Exhibitions* which are composed of *Artworks* which themselves are placed in *Rooms*. Artworks produced by an *Artist* in a certain *Historical Period* can have anonymous *Comments* (added by visitors of a Web site) and *Evaluations* done by *Experts*. All *Artworks* have an *Owner* and, for simplicity, we assume the owner to be a *Person*. An *Artwork* in restoration also knows the *Restorer*'s identity. The *Auction Manager* can put an *Artwork* in *Auction* and knows the *Owner* in order to eventually negotiate the conditions of the *Auction*. Though not included in the diagram, Auctions for an Artwork are organized in a particular date and place. An *Artwork* can be in *Exhibition* and at the same time being scheduled for an *Auction*.

The model above has basically two problems:

- The responsibilities (informational and behavioral) in class *Artwork* are many, as artworks are objects that collaborate with many others in the system. Some of those collaborations are only specific to a certain collaboration context. For example, when being scheduled for an auction, an artwork has to "acquire" knowledge about the schedule (knowledge that by the way is irrelevant when it is only an artwork in exhibition). In the same way, the concept of current price of an artwork (e.g., as the answer of a method with that name) makes non-sense for most artworks in exhibition which don't have a price. The same problem appears with artworks in restoration. The notion of date only makes sense in the specific case that an artwork is in restoration. Notice that while some properties of an artwork are intrinsic to the artwork (e.g., its name, the date in which it was created, etc), other properties arise when the artwork participates in some relationship, such as the relationship with an auction or a restorer.
- Though less important, the modeling of *Owner*, *Artist*, and *Expert* as sub-classes of *Person* also show another face of the previous problem. If the same person is the artist which painted an artwork and the owner of the same (or another) artwork, the model shown above doesn't work fine. Once again, the intrinsic properties of a person (e.g., her name) are mixed with those which pertain to a relationship in which the person is involved, such as being the creator of an artwork.

A naïve solution to the first problem above could be to define a class hierarchy of *Artwork* classes, to model the variations (e.g., *ArtworkInAuction*, *ArtworkInRestoration*). This approach does not work as the same artwork might have to belong to different classes (e.g., when it is being auctioned and in restoration) or even worse it might have to "change" its base class frequently. A more subtle solution could be to consider "in restoration" and "in auction" as possible states of an object and try to model the differences according to the state. In this case, even if we use a modular solution such as the State pattern [15], we have the problem of an artwork being in multiple states at the same time (situation not considered in the State pattern).

The essence of the above problems lies in their nature. In fact, the problem is that the same object needs to exhibit different properties (behaviors or information) according to the relationship in which it is involved (or different behaviors according to the subject which sends it a message), and this is only possible if we can separate the intrinsic properties from those properties which depends on a relationship. In the object-oriented paradigm, there are no "native" constructs that allow separating the intrinsic properties of an object from those which depend on the context in which the object is being used. As shown in the next sub-section, this is a base problem of the paradigm which manifests itself in slightly different ways during conceptual and navigational design.

As a simple example, when an artwork is being accessed from an auction, it must behave as expected by the auction. Meanwhile, when it is accessed from an exhibition, it behaves "just" as a plain artwork. Even though this problem exceeds the domain of Web applications (it is a general software design problem), the fact that many different Web applications are usually built from the same conceptual model, e.g., for accommodating to different audiences (as in [11]), makes the problem outstanding in this domain, as we need to emphasize modularity and simplify evolution. In other words, if our conceptual model is built to be an integrated domain model and therefore its classes need to provide the above flexibility, we need to solve these, say, micro-architectural problems.

Notice that if a new application arises in the model (e.g. a new type of audience not previously foreseen), and the navigation model requires some “new” operations to be performed in the domain model, then the (shared) domain model must evolve to accommodate to this new situation; this evolution is only possible if we are able to ensure modularity, when application classes must support new sets of behaviors.

A design solution to this family of problems is to use the Decorator pattern [15] to manage object behavioral and structural extensions. By keeping the intrinsic properties in the base class, we separate the additional (relation-oriented) properties and specify them in the corresponding decorator which is composed with the base object.

However, while the Decorator provides a hint to the solution, it has two problems: first, it provides a solution of a somewhat low-level nature (more targeted to implementation) and consequently might obscure the conceptual model; second, its intent is too broad (which is reasonable given that it belongs to the concern of design and not just modeling) and needs to be refined to the target problems described before.

What we need is a higher-level concept and a corresponding notation to express the same idea. As we will show from Section 3 on, the role concept (and its realization using object composition) gives an elegant conceptual tool to solve these kinds of problems.

2.1.2 Navigational Design

Being Web applications a particular kind of hypermedia software[a], navigational design aims at defining which hypermedia objects the user will perceive and how he will navigate them; hypermedia primitives such as nodes, links and indexes are used in this modeling activity. Nodes and links are generally defined in terms of application objects and relationships, using a mapping mechanism for all classes that are relevant for navigation. As different types of users will have different tasks to perform using the system, a separate navigation model might be specified for each type of user, e.g., administrator, art lover, bidder etc. Following our example on the museum, a simplified navigation model showing the most important classes of nodes and links is presented in Figure 2. We assume an art lover as the user of the application which results from this navigation model.

The navigation model (Figure 2) shows the result of a first step in the navigation modeling process. It consists of the visualization of navigation nodes and navigation links, showing the navigation paths that the user can follow while browsing through the museum application. The navigation model also comprises the so called access structures. Access structures are, e.g., menus or collections, indexes, and guided tours. The names of the modeling elements used for these access

[a] Some Web applications might not follow exactly the hypermedia paradigm. Even though many of the theoretical ideas exposed in the paper can be applied to a broader range of applications (See [47]) we will limit our explanation to Web applications in which there is some navigational behavior, i.e. in which nodes are connected with links.

structures as well as their graphical representation vary from method to method. For example, UWE divides the navigation design stage in two sub-stages, one is completely devoted to access structures. Thus, for the sake of simplicity, we will not present here a complete navigation model.

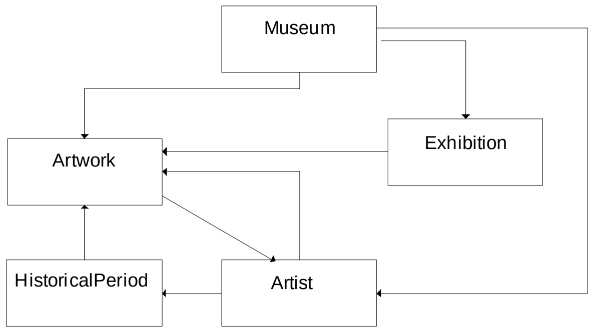
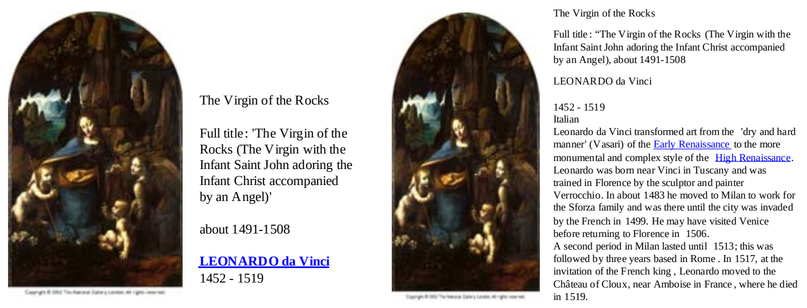


Figure 2 Sketch of the navigation model for the museum.

As we show in Figure 2, we can access an artwork node by following different paths: we can see artworks in an exhibition, or navigate to an artwork from the artist node. Similarly, we can see an artwork after examining the historical period in which it was created. We can access an artist from an artwork and also from the museum.

This simple and naïve diagram, however, poses a problem. A well-designed Web application should provide information taking into account the actual navigation path. In other words, when reaching an artwork from the exhibition in which it is included, it would be nice to read a short biography of the artist; meanwhile, while navigating to the same artwork from the artist node, the biography does not make sense. Similarly, when accessing the artwork from the historical period node, we would like to read some additional explanation on the context in which it was created and, perhaps, have access to other links (not meaningful in other navigation paths). In the left of Figure 3, we see the artwork when accessed from the artist node; it does not show any information on him. In the right, the same artwork is accessed from the exhibition node and presents a brief summary on Leonardo’s life and a link to Leonardo’s node.



The Virgin of the Rocks

Full title: 'The Virgin of the Rocks (The Virgin with the Infant Saint John adoring the Infant Christ accompanied by an Angel)'

about 1491-1508

LEONARDO da Vinci
1452 - 1519

The Virgin of the Rocks

Full title: “The Virgin of the Rocks (The Virgin with the Infant Saint John adoring the Infant Christ accompanied by an Angel), about 1491-1508

LEONARDO da Vinci

1452 - 1519
Italian
Leonardo da Vinci transformed art from the 'dry and hard manner' (Vasari) of the Early Renaissance to the more monumental and complex style of the High Renaissance. Leonardo was born near Vinci in Tuscany and was trained in Florence by the sculptor and painter Verrocchio. In about 1483 he moved to Milan to work for the Sforza family and was there until the city was invaded by the French in 1499. He may have visited Venice before returning to Florence in 1506.
A second period in Milan lasted until 1513; this was followed by three years based in Rome . In 1517, at the invitation of the French king , Leonardo moved to the Château of Cloux, near Amboise in France , where he died in 1519.

Figure 3 Navigating to an artwork from different paths.

Notice that the problem above is somewhat equivalent to the one arising in the conceptual design activity: we want that some nodes “behave” differently according to the navigation path, in this case by showing different information; more generally, we want the behavior of the node to change according to the incoming link.

One possible solution could be to add a decision choice when the node is opened (thus solving the problem “procedurally”), which will tangle the decision into the code for opening the node. Another solution is to define different Artwork nodes in Figure 2 (each one being a view on the *Artwork* class in Figure 1). This solution forces us to repeat information on each view with eventual maintenance problems. Eventually, we could also build a class hierarchy of *Artworks*, but this solution introduces a spurious specialization criteria, which may conflict with other future extensions.

Similarly to the problems solved in the conceptual model, the solution can be found by using a compositional role-based approach. As we show in the following sections, this approach is easily modeled using object-oriented roles.

2.2 Summary of Design Problems

To summarize the discussion above, object-oriented Web Engineering approaches fail to address the following problems:

During conceptual design of the Web application domain:

- It is not easy to separate different behavioral concerns. For example when the same object (e.g., an artwork) should behave differently according to the client object (i.e., the object which sends the corresponding message). An artwork which is in an auction should have an interface which includes messages to add a new bid or to get the best bid; an artwork in restoration does not need these behaviors but others.
- As a consequence of the above problem, how to indicate that certain information (e.g., an attribute or relationship) of the object is only meaningful in the context of a specific context of collaboration. For example, the date of an auction corresponding to an artwork makes only sense if the artwork has been tagged to be auctioned. In this context, the artwork might also exhibit relationships to the auction house, etc. Specifying different *Artwork* sub-classes (e.g., *In Auction*, *In Restoration*) is not a solution, as it is far too rigid. If the same artwork can be in two different situations (e.g., In Restoration and later in Auction), a class hierarchy prevents us to assign different classes to the same object.

During navigation design:

- How to cleanly and compactly indicate that a node (such an artwork) might exhibit different information when accessed using different paths. For example how to specify that when navigating from the Historical Period, the artwork node shows some attributes which are not meaningful for other paths.
- How to indicate that a node provides links which depend on the path in which we have accessed a node. For example, when accessing the artwork node from the HistoricalPeriod

node, we might wish to define other outgoing links that allows further exploration of historical issues related with the artwork.

The core of these problems lies in the nature of relationships between objects in the object-oriented paradigm. Objects are usually agnostic with respect to the subjects which send them messages (usually known as clients): i.e., objects always react in the same way regardless which object sent the message. At the navigational level, the problem seems different (as it seems that no message exchange is involved), but it is essentially the same.

From the modeling point of view, we need a way to indicate the previously intended variations. In the following section, we introduce the concept of object roles and, in further sections, we show how we use it to solve the previously illustrated problems.

3 The Role Concept and its Use in Web Engineering

There exist a couple of different definitions and semantics of the "role" concept in computer science [47]. The "role" concept has been used in different levels of abstractions: as a primitive in conceptual modeling [47, 48 or 49], as a tool for improving design expressions [38, 39 or 40] and even for extending programming languages and environments like Smalltalk [30]. The term "role" is used by analogy to the theater where an actor "plays a role", thereby takes actions which characterize the role. For an object, all definitions share the same intent: to let the object have different properties and behavior in different situations at different times. In this section, we clarify which is the definition of the role concept our approach is based on. Insofar as we extend object-oriented Web engineering methods, we adopt the same view on roles as in the object-oriented programming world [30]. We first introduce informally the role concept as considered in this view. Thereafter, we precise the corresponding definitions related to roles. Finally, we argue on the visual notation to use by a brief comparison with other definitions of roles.

3.1 Background and Definition

An object may have various behaviors and properties at different times according to the set of objects it is involved with in some well identified situation. We say that the object plays a specific role in this situation. For example, an artwork, as an object, can play two roles: being "in exhibition" in a museum and being "scheduled for an auction". However, objects of different types may play the same role. For example, an artwork object and a manuscript may be "in exhibition" (in the same exhibition or in different exhibitions).

Fundamental for understanding the relationships between roles and objects is the distinction between *natural types* and *roles*, originally introduced by Sowa [46].

"A natural type is a semantically rigid and non-founded type insofar as an entity that has the type cannot stop being of the type without loosing its identity and does not depend on any collaboration". As an example, consider the Type *Artwork*: an object of type artwork exists independently of a role it can play in relationships with objects of other types such as an exhibition or an auction. It is not possible that an object end being an artwork without loosing its object identity. On the contrary, it can stop playing the role of being scheduled for an auction at some time without ending to be an artwork.

On the contrary, “a role type is a founded and semantically non-rigid type insofar as it characterizes an entity by some role it plays in relationship to another entity or other entities, and if left, does not give up identity of entities”. An example of role type is “scheduled for an auction”. It characterizes specific properties and behavior that an artwork object or a manuscript acquires as soon as the auction manager puts it in auction. These characteristics are lost when that auction on the object is finished. Thereby, a role type characterizes the dynamic state of an entity when it is involved in some collaboration with others.

Let us remind in a more abstract way the definitions in [30] that we adopt. The role type *x* of an object *y* refers to those additional properties of *y* (attributes and behaviors) that are critical for the object in a particular kind of collaboration, i.e., when interacting with other objects in a certain context. We say that *y* is playing the role *x*. Those properties of *y* that exist independently of any role are called *intrinsic properties*; they are defined by the *natural type*; meanwhile the properties that the object “acquires” when playing a role are called *extrinsic*. More formally, in an object-oriented world, both *x* and *y* may be defined by classes; a role type x defined by a class *X* thus specifies the abstract structure of instantiations of role *x* that belong to *X*, i.e., the extrinsic properties of y. Meanwhile, the intrinsic properties of y are modeled by its natural type defined as a class Y. A role instance is dynamically bound to an (intrinsic) object when it begins playing the role. Such role instance has a specific state and a specific behavior.

3.2 Our Notation for Roles

It is not our aim to describe all different notions of role (the reader can refer to [47]) and their corresponding notations in the literature; however, in order to explain the notation we choose, we can distinguish between two broad groups of works.

The first group is represented by the data models that derive from the popular entity-relationship (ER) model, the most representative being the object-role modeling technique (ORM). ORM pictures the world in terms of objects (entities or values) that play roles in relationships [22]. Objects do not have attributes; instead they are “built” according to the relationships in which an object is involved and potentially each object’s attribute is defined in terms of a role. In ORM, roles improve typical ER relationships by indicating with a name, the role of each participating object. ORM provides a rich notation, a set of primitives for indicating constraints of relationships, roles and objects and a set of heuristics for deriving relational tables from a conceptual model. ORM does not provide means to distinguish between intrinsic and extrinsic attributes; besides, ORM is a data model, which means that its aim is describing objects’ data more than objects’ behavior.

The second group encompasses object-oriented approaches which basically consider role as “enrichment” of objects while they interact with others in a similar way as we do (see for example [31). Extreme applications of this same idea are [40] and [37] where role models are used as higher-level abstractions than class models. In both cases, a role is defined as a set of behaviors and a role model describes a set of interacting roles. While in [38] and [39] this approach is used to specify design patterns that will be later instantiated by making application classes play the corresponding roles, in [37] a role model is defined in the early steps of a development methodology (see Figure 4.b).

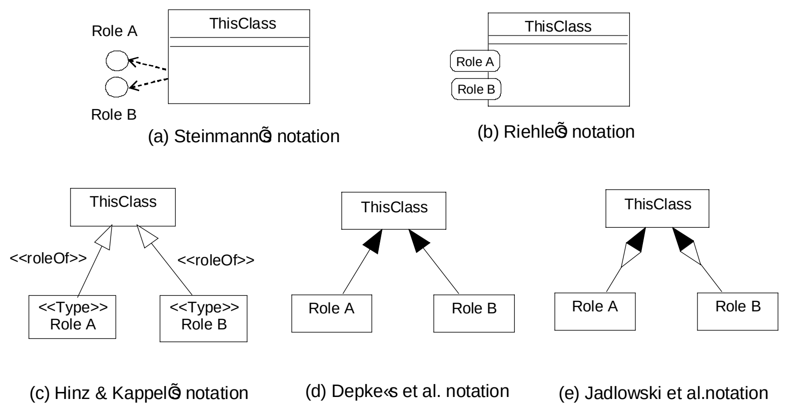


Figure 4 Notations for roles.

Approaches such as [23], [10], and [24] incorporate the role concept in the design armory by a heavyweight extension of the UML meta-model. They focus on the relationship between a base class and a set of classes that define the roles that class can play. Each of these approaches defines a "role of" relationship, which can be applied at class and object level. Hinz and Kappel [23] following [19] introduce a UML *roleOf* relationship between classes (see Figure 4.a). It is represented as a stereotyped inheritance relationship at class level, but it has the semantic of inheritance on instance level (see Figure 4.c). [10] defines the *role-of* relationship as a subtype of the UML metaclass *association*, but combining characteristics of generalization and composition. Contrary to a generalization, they suggest to depict the *role-of* relationship also in object diagrams (see Figure 4.d). Similarly, Jodlowski et al. [24] propose a notation that visually resembles both the composition (filled diamond) and the inheritance (triangular arrow) notation of the UML (see Figure 4.e) stressing that the role relationship is characterized by the mix of their properties. They also use the same symbol at class and object level.

To summarize, insofar as we extend object-oriented Web application design methods by considering role in a similar way of [30], we chose the Riehle's notation (see Figure 4.b). "ThisClass" is the class which defines the natural type of an entity. "Role A" defines A as the type or class of a role an entity of class ThisClass can play in the situation defined by a diagram where ThisClass appears with a possible "Role A". Let us remark that for UWE which is based on UML, we will introduce in section 5 a new notation in order to eliminate confusion introduced by notations represented in figures 3.a, 3.c, 3.d, and 3.e, as explained above.

3.3 Using Roles in the Web Engineering Life-cycle

3.3.1 Conceptual Modeling

From previous definitions, it is clear that the role concept improves the expressivity of object-oriented conceptual models by clearly separating and indicating those objects' features (structure and behavior) which depend on the context of invocation (i.e., the sender of the message) from the intrinsic object properties. Introducing roles in conceptual modeling as first class entities makes it possible to solve problems such as those mentioned in section 2. For example, insofar as objects (as instances) might be involved in many relationships, an object y may play several roles (e.g., x and z) at the same time. It can even play the same role x twice (in different places). For instance, an artwork object may be in exhibition in two different museums (for example it may be exhibited at different times of the day). Besides, a role type can be "assigned" to different natural types (for example polymorphic types) or even to other role types. For example, a same *Person* y can play the role of *Artist* as the person involved in the relationship "creator of an artwork" and might additionally play the role of *Owner* as long as she has not sold it.

It will now be possible to distinguish explicitly extrinsic properties (attributes and behaviors) that objects acquire by playing some role when they are involved in some situation, from their intrinsic properties as defined by their natural type which remains the same whatever be their situations. By doing so, we will be able to model application behavior and navigation precisely and solve the problems mentioned in section 2-2: particularly, express which properties and behaviors an object has when being involved in a relationship or specifically when a collaborating object communicates with it. For example, an artwork in an auction will have, besides its intrinsic properties, the initial bid for that artwork (property which does not make sense in another context).

We show a sketch of the corresponding schema in Figure 5. A similar schema can be defined to improve the definition of the different roles a person can play: *Owner*, *Artist* and *Expert*. In Figure 5, we show the two role types: *InAuction* and *InRestoration* which contain the extrinsic information and behaviors which artworks must include when involved in relationships which need to treat them accordingly to the role. The *Artwork* class exhibits those two roles (denoted with the syntax of Figure 4.b); an association which has a role as the target indicates that the artwork will be playing that role, which means that its structure and behavior is enriched with the structure and behavior indicated in the role. Associations which have the origin in a role are only visible when the object is playing that role. An additional advantage of using this approach is that we can add new roles to the *Artwork* class without having to modify Artwork.

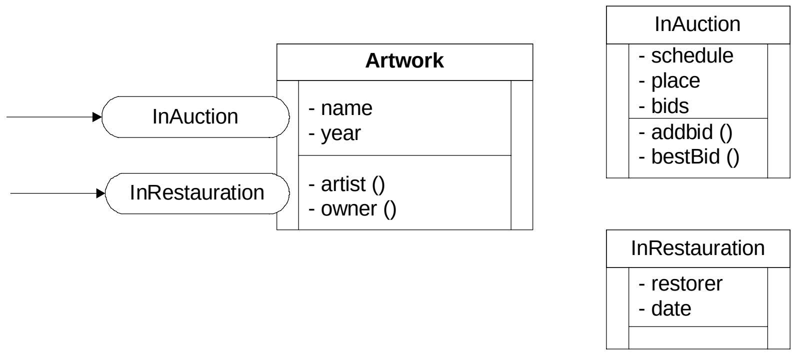


Figure 5 Different roles for the *Artwork* object.

As conceptual schemata in object-oriented Web design models follows well known practices in the object-oriented paradigm, the reader is referred to [30] for further issues such as role composition and inheritance. A comprehensive approach for dealing with roles and role models in object-oriented systems can be found in [3].

3.3.2 Navigational Design

The first way of using roles during navigational design is to profit from role definitions in the conceptual model to derive nodes. For example, we can define a navigation class *Owner* whose information is derived from the role (class) *Owner* associated to persons. Notice that owner nodes will contain attributes taken from both the intrinsic person object and from the corresponding role object.

Another simple way to use roles defined during conceptual modeling arises when building an application for a particular user profile, e.g., the *Auction administrator*. In this case, we will design artwork nodes such that they provide interfaces for those behaviors that correspond to the auction's role as shown in Figure 5. In this case, corresponding nodes will be built as views on the *InAuction*'s role of the *Artwork* class, as this application only needs to consider artworks in auction. This previous example is interesting to note, especially when the audience of the Web application (the auction administrator) coincides with one specific role defined during conceptual modeling.

However, these two uses of roles do not increase the expressive power of navigation design. They are just a smart way to profit from the improvement of information and expressivity in the conceptual design (i.e., the addition of object roles) to create new class of nodes. Of course, this improves the overall application functionality.

Being hypermedia (and as a consequence navigation design) a discipline in which relationships (materialized as links) constitute an important feature, it is clear that the role concept can be applied in a simple and intuitive way to allow nodes to exhibit different characteristics (such as displayed information or outgoing links) when accessed with different links, i.e., when they are related with different nodes. Particularly, roles allow to:

- Express that a same object displays different features according to the navigation path followed by the reader. For instance, it will be possible to express that a short biography of the artist should be displayed when accessing an artwork from the exhibition. Meanwhile, this same biography does not make sense when accessing the same artwork from the artist.
- Express that a node provides links that depend on the context in which it is accessed. For example, when accessed from the historical period node, an artwork node offers outgoing links for further exploration of historical issues related to the artwork which are not offered when accessing the artwork from the auction node.

In our example, an application for art lovers and focusing on artwork nodes (defined from the conceptual class *Artwork*), we can decide whether we want the artwork to show different attributes or behaviors (as shown in Figure 6) by analyzing the different incoming link types. For example, when an artwork is accessed from the exhibition node, we may want to include the biography of the author and some comments related with the place of the artwork in relationship with the exhibition; we might also provide a link to other relevant artworks in the same exhibition. Meanwhile, when we navigate to the artwork from the artist page, we might not want to show the author biography and just show the “bare” artwork. More generally, we could eventually define different role types for each incoming link type of a node class. In Figure 6, the *InExhibition* role type contains an attribute and two outgoing links (to *Artist* and to *Artwork*) which indicate that when an artwork is accessed from the exhibition page, it will exhibit an additional attribute and these links.

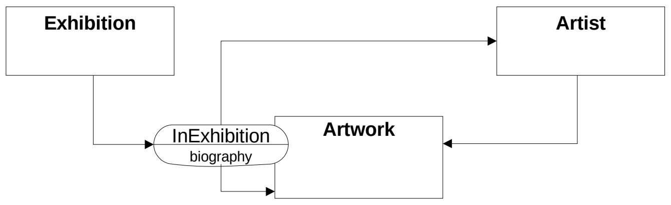


Figure 6: Improving navigation design using roles.

3.3.3 Conceptual vs Navigational Roles

The use of roles in conceptual and navigational design follows slightly different rationales as explained before. We analyze here those situations in which we combine roles in the conceptual and navigational models.

The first situation arises when a class is enriched with roles in the conceptual model, but not in the navigation model. This is the case both for classes which are not useful during navigation (e.g. *Person* and its roles), or for purely behavioral roles, such as an artwork in auction. In this case, the conceptual role is necessary but there is no equivalent role in navigational design.

The second situation appears when a conceptual class is not enriched by a role, but the corresponding navigational class is enriched by a role. This is the case with artworks in an exhibition. While a role is not necessary to express additional attributes or behaviors in the conceptual model, we use it in the navigational model to allow different information to be provided during navigation.

Finally, there are situations in which we can enrich roles in the conceptual model with roles in the navigation model. For example, suppose the Web application built for supporting auctions (not just exploration of artworks). In this application, the navigation class *Artwork* is built from the *InAuction* role class, as we may want to provide users with new facilities (e.g. to add a bid). However, an artwork node might also exhibit different information according to the way we reach it. For example, when we access the artwork from an artist, we may want to provide links to other artworks of the same artist, which will be auctioned shortly; similarly, when accessing the artwork from the index of open auctions, we may want to have links to other artworks in auction at the same time. In both cases, we are finally refining the conceptual role *InAuction*, with different navigational roles (for different ways of accessing artworks).

3.4 Discussion

An interesting issue to be discussed is whether the role construct is necessary in the Web design methods armory; in other words, could we solve the same problems addressed here, without using roles?

The literature on object-oriented design has extensively discussed the previously mentioned problems and their variants [15, 30]. Examples in the literature abound with respect to the need to support the kind of variations in objects' features we have previously discussed.

A first conclusion stated before is that the base object-oriented primitives and abstraction mechanisms are not enough to design objects whose properties must vary (dynamically) according to the collaboration context. In other words, it is not possible to easily separate the intrinsic properties of an object from those properties which arise during the collaboration with another object.

Inheritance might be used in some cases but the introduction of sub-classes in these situations introduces additional problems. As discussed before, defining sub-classes of *Artwork* to indicate that an artwork is in restoration or in an auction is a bad solution because one object can not change its class, nor it can belong to two different classes at the same time, for example if the artwork is both in restoration and in auction.

Similarly, defining sub-classes of the *Artwork* node class to indicate differences according to the incoming link pollutes the class hierarchy which might contain other sub-classes defined with a better rationale like Painting, Sculpture, etc. Another alternative could be to define different views on class *Artwork* to define subtly different node classes. However, once again, we neglect to separate intrinsic from other node's attributes and outgoing links (those that pertain to each possible incoming link). In this way, we have to repeat the definition of the intrinsic properties in each view, thus compromising modularity.

Good object-oriented designers have solved this kind of problems once and again using design patterns. We mentioned before the Decorator pattern [15] which aims to enrich a class with new behaviors in an instance basis (e.g., different class instances might exhibit different behaviors).

Closer to our approach, the Role Object pattern [4], specializes the decorator by focusing to a more specific problem and similar to the work of [30] analyze different role variants. We believe that this is a better solution though the diagrams in [41] are quite complex; they are targeted to low-level design and, thus, they are not supposed to be used during the modeling stage. If designers solely rely on existing modeling and abstraction mechanisms (e.g., in the object-oriented paradigm), they might end with difficulties to read diagrams in which high level decisions are cluttered with spurious compositions, aimed for example to express the needed decorations to specify the intended functionality [15].

Instead, we propose to enrich existing object-oriented Web modeling approaches with a new primitive: the role construct; therefore, we add expressivity with a very low cost in diagram complexity as we show in Section 4.

The use of the role concept has an important impact in the modularity of the design structures and, therefore, allows improving evolution, maintenance, and reuse. First, by avoiding to create spurious sub-classes to accommodate structural and behavioral variances, we keep base classes (those that implement natural types) concise and focused to the concept. By separating role classes, we improve reuse of the natural types, as they are just concerned with the core ideas of the type, and we also improve reuse of the role classes, as they are not bound to the natural type, but evolve independently. For example, the role *InAuction* (previously described) could be used in applications which do not involve artworks but other kinds of material. This shouldn't have been possible if

InAuction were defined as a sub-class of *Artworks*. Regarding evolution, we can create sub-classes of *Artwork* regardless of their involvement in auctions or exhibitions; in other words, the engineering of artworks is not coupled with a particular relationship in which artworks are involved. In the same way, by separating the core concepts of a node (e.g. artwork) from the contexts in which it is accessed, we can add new contexts without caring about the properties of the core node abstraction.

4 Putting Roles to Work in Web Engineering Methods

The role abstraction mechanism complements existing object-oriented modeling concepts, mainly the class concept. The sole introduction of this concept in the modeling armory gives us the possibility to think about applications in a higher-level way, by indicating how the behaviors of objects change in the context of a particular collaboration. However, the introduction of the role concept in existing methods has to be done in a seamless way, in order to maintain the coherence of the corresponding notation. In this section, we show how to improve two existing Web engineering methods with roles.

We have chosen OOHDM and UWE for some simple reasons. First, they are object-oriented (i.e., their base abstractions are collaborating objects); thereby they can easily benefit from the motivations behind this work. They are also well recognized in the community. Second, and not less important, two of the authors of this paper have a huge experience with the engineering of these methods and, therefore, they can better use their own extension philosophy and mechanisms; however, the discussion below can and should be applied to other methods. As introducing a new concept in an existing methodology involves much more than improving a notation, the sub-sections below are just the starting point for a much wider discussion in the Web Engineering community. For the sake of comprehension, we describe each method separately and as a last sub-section we discuss implementation issues. We show that not only roles help to solve the problems mentioned in section 2 but also greatly improve these methods.

4.1 Introducing Roles in OOHDM

In OOHDM, the conceptual model is described using a notation similar to UML. Therefore, and as discussed in section 3 and illustrated in Figure 5, we can introduce roles in the conceptual model by using the notation in Figure 4.b. Roles (or role attributes) in the conceptual model can easily be mapped into nodes (respectively, node attributes) by using the viewing notation of OOHDM [43, 44, or 45]. We no longer detail the application of roles in the conceptual model as discussed in Section 4.1. Rather we focus on a more original contribution, the introduction of “pure” navigational roles. In OOHDM, the navigational model represents a view on the conceptual model for a specific user profile. The navigational structure of the application is specified with a navigational schema and a navigational contexts schema. The navigational schema shows the node and link classes of the application; their semantics are the usual in hypermedia applications; nodes contain anchors for links.

In Figure 7, we show, according to the OOHDM notation, part of the navigational schema for the museum example, considering as the application’s audience art lovers exploring the museum. Using plain OOHDM we can not express any difference in the contents of an *Artwork* node when we access it from an *Exhibition* or from an *Artist* node.

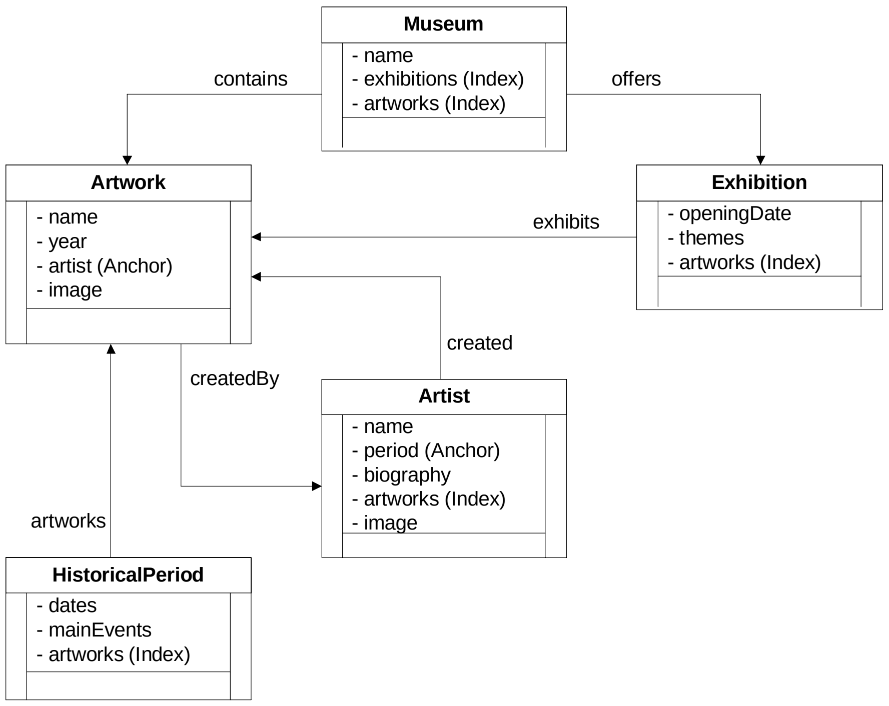


Figure 7 Navigation schema in the museum example.

In Figure 8 meanwhile, we show the navigational schema of Figure 7, but now exhibiting the different roles that nodes can play according to the corresponding navigational path.

Artwork nodes exhibit two roles depending on whether they are accessed from *Exhibition* or *HistoricalPeriod*; analogously, when we access a historical period from an artist, it includes a short contextual description of the artist in the period. Notice that in this last case, for each artist we will get a slightly different description of the period. Notice that, by decoupling this description from the period itself we manage to provide different information when navigating from an artwork (in the exhibition role).

An original feature of OOHDM is the introduction of the notion of *navigational context* defined as a set of nodes sharing some property, e.g., artworks of an artist, artists that lived in a certain period, artworks in an auction, etc. When a node is navigated in a particular context, it might exhibit specific features, which are specified by using InContext classes, which enrich the corresponding node class. As contexts are sets of nodes, the kind of information added by InContext classes relates with improving access to the set; for example links pointing to the following or previous node in the set can be added. For example, Artworks could be organized by date and displayed chronologically, or by technique and, given one technique, all artworks that correspond to that technique could be displayed sequentially. Since nodes involved in a navigational context play a specific role when being accessed in the context, it is natural to use now our role-based notation for representing different features of those nodes in the context. For example, in Figure 8, we would have two new roles for Artwork: ByTechnique and ByDate; each role will have two anchors (Next and Previous) and corresponding links to allow sequential navigation in the set.

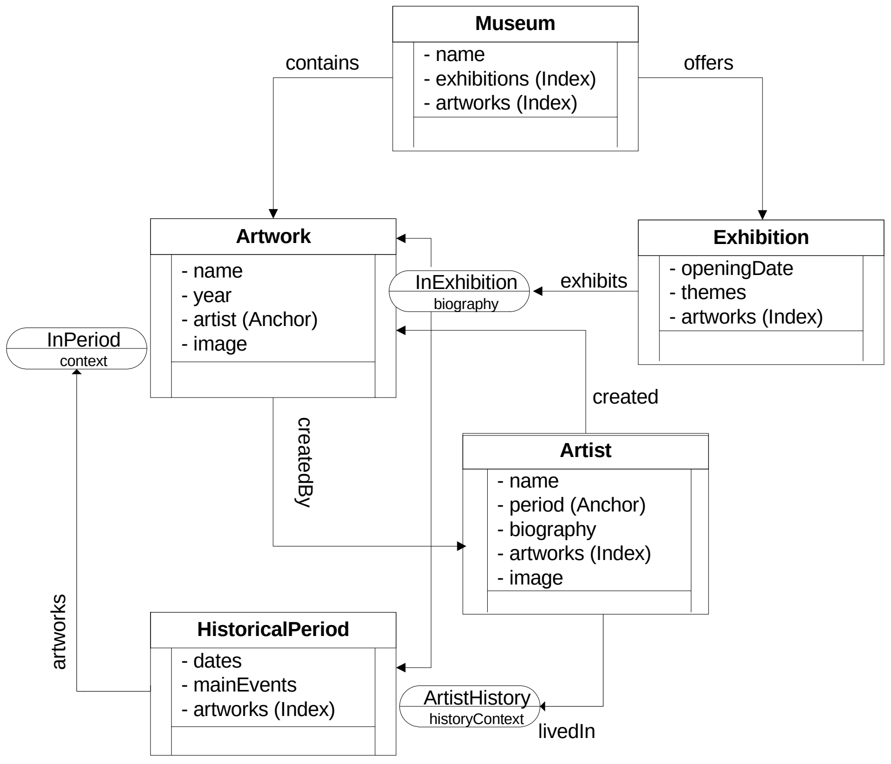


Figure 8 Improving the OOHDM navigational schema with roles.

To summarize, roles improve greatly the old OOHDM InContext classes in two different ways: first, their aim is more general, consequently one can use roles for solving problems which do not involve sets, and second, the semantic of roles is of a higher level of abstraction compared with InContext classes. While roles are a modeling construct, InContext classes are an opportunistic design solution. The addition of the role concept in OOHDM does not imply additional diagram complexity. Figure 8 is understandable by any OOHDM designer accustomed to using InContext classes. Moreover, being of a higher-level of abstraction, it is easier to introduce to non-expert users, as it clearly transmits its intent.

4.2 Introducing Roles in UWE

The UWE methodology [27] is also an object-oriented approach but purely based on the standard UML, i.e., the notation and diagrams of UWE are restricted to those provided by UML. The UWE metamodel is defined as a conservative extension of the UML 2.0 metamodel [28]. Conservative means that the model elements of the UML metamodel are not modified and new elements are related by inheritance. This metamodel is the basis for the UWE notation, which is defined as a light weighted UML profile using the UML extension mechanisms: stereotypes, tagged values, and OCL constraints [34]. The advantage of this approach is that UWE can benefit from all tools that support UML. For the conceptual model of the content of the Web application UWE uses plain UML primitives as shown in the museum example (Figure 1). The UWE profile introduces some specific modeling elements for navigation and presentation. In particular, the navigation model is represented by a stereotyped class diagram, including the classes of those objects of the content model which can be visited during navigation (Museum, Artwork, Exhibition, Artist and Historical Period). The «navigation class» and the «navigation link» stereotypes are used to model nodes and links. Relationships with content classes are expressed using the UML Object Constraint Language (OCL) [34]. Other stereotypes such as «menu», «index», «query» and «guided tour» are used for further refining the navigation model. An index for example allows direct access to instances of a navigation class. This is modeled in UWE by a composite object, which contains an arbitrary number of index items. Each index item is in turn an object which has a name that identifies the instance and owns a link to an instance of a navigation class. Figure 10 and Figure 11 show two alternatives for the use of the icons of UWE stereotypes.

In this section, we focus on the use of roles in the navigation model of UWE. Depending on the context in which a navigation node is navigated, such a navigation node can play different roles. To play a different role implies that different features and different behaviors of the node can be perceived by the user or shown to the user. The context may be defined by the functionality the navigation node has to provide in the context or it is given by the role of the user who is navigating. In UWE, roles are defined through a directed relationship *role of* between a base class and a role class with the semantics that objects of the role classes inherit their non-role specific attributes and behavior from the base class and role class has specific features defined by the role class. In order to represent the role semantic graphically, the UWE armory of elements are augmented with an extension of the UML modeling element *generalization* (see Figure 9). We choose the name *role of* for this extended generalization similarly as already proposed by [23, 10] although the semantics of this modeling is slightly different, mainly in order to allow a light weighted extension of the UML [34] metamodel similarly to the UWE extension [28].

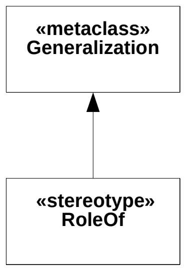


Figure 9 UML extension including a «role of» modeling element.

This way we can model a navigation node, i.e., navigation class, a menu or an access primitive in a navigation model from different viewpoints, depending on the role it plays in the navigational context. Applying the *role of* relationship repeatedly for a role class taking the position of a base class leads to the construction of role hierarchies. The lifetime of the instances of a role class is directly dependent of the lifetime of the instance of the base class that is connected to by the corresponding *role of* link. Regarding the visibility of features the transitivity of the inheritance relationship implies that all instances of a role class may access to all features of the base class or all features of the role classes which are in a higher level of the role hierarchy.

In Figure 10, we show the specification of a set of classes for the different roles the navigation class *Artwork* plays in our example application. These role classes are *InPeriod* and *InExhibition.* They are related to the base class Artwork through the *role of* generalization.

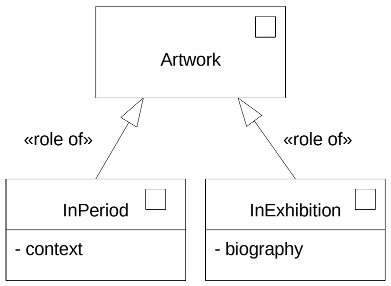


Figure 10 Roles for navigation class *Artwork*.

In Figure 11, we show part of the basic UWE navigation model for our running example. Note that the base class *Artwork* and each of the role classes *InExhibition* and *InPeriod* are target of different navigation links. Figure 12 shows the UWE navigation model enriched with indexes.

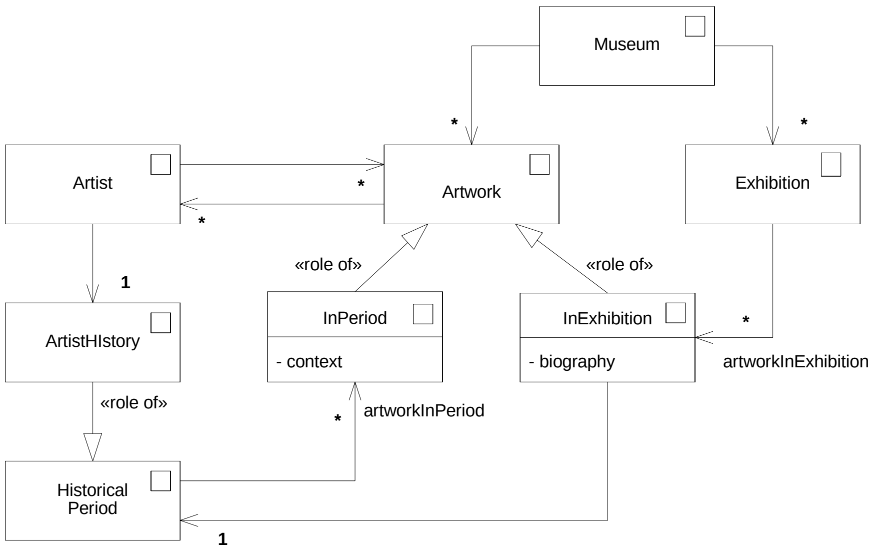


Figure 11 UWE navigation model improved with roles.

The benefits of using roles in UWE are twofold: first we can use roles in both the content and navigational model keeping the notation UML-compliant; second, the use of roles in the navigation model allows improving the method, as expressed in Section 3.

With roles in the modeling armory, UWE can express differences in the navigation path. In Figure 12, the *ArtworkInExhibitionIndex* points to specific roles of the *Artwork* navigational class, named *InExhibition*. Notice that the same kind of navigational behavior is represented by every role of a navigational class referenced by an index, thus introducing set-based navigation as in OOHDM contexts.

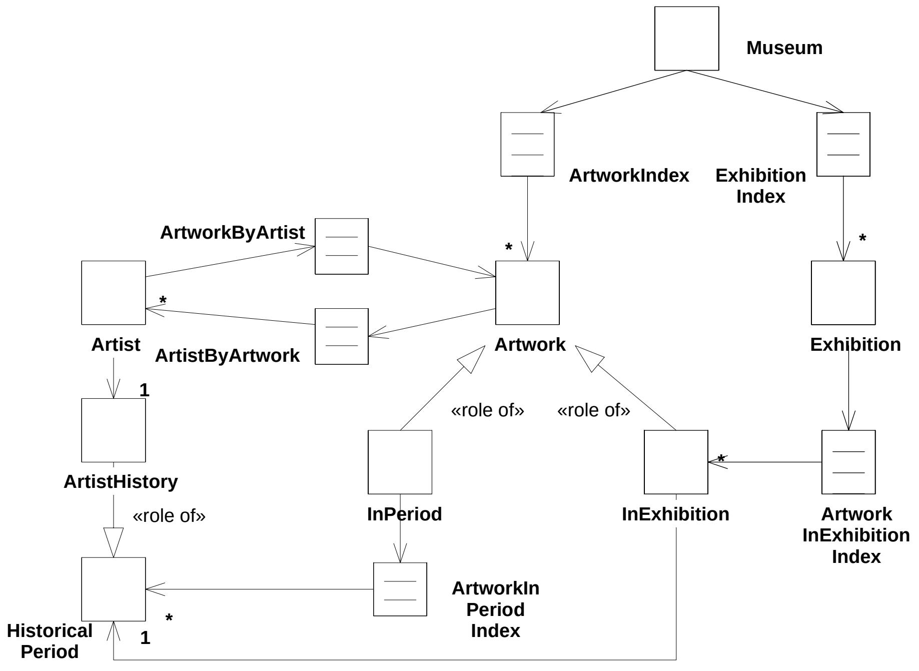


Figure 12 UWE navigation model improved with indexes (UWE stereotypes represented by icons).

One interesting issue of this kind of extension (which closely follows the philosophy of UWE with respect to UML compliance) is that the role-enhanced diagrams are not complex. The addition of new functionality is done without cost, as the notation is similar to other possible UML light weighted extensions (i.e. supported with the UML extension mechanisms).

4.3 Further Issues

We have also used roles for other different purposes in which some semantic flexibility (not provided by the basic object-oriented primitives) is necessary:

- to deal with real-world objects in physical hypermedia applications [20], those mobile applications in which physical and digital objects are combined [18].
- to implement concern-driven navigation structures [17].

While explaining the first point requires introducing aspects of mobility, not interesting for this paper, we summarize the second here.

As previously shown, the introduction of roles in the navigation modeling activity allows to express in a compact way the information which a node must exhibit according to the incoming link. The examples above demonstrate that, by using roles, one can eventually show different information according to the path the user is following to reach the node. In [17], we have presented the idea of concern-driven navigation, a way to structure the navigation model according to the theme or concern the user is interested in. A simple example in the context of the same museum application could be the possibility of accessing artworks emphasizing technical aspects (materials used for the artwork), or having access to the artwork's history (the places where it was, the museums in which it was exhibited, its previous owners, etc). Notice that these two simple examples might involve a set of brand new classes in the previous schemata, e.g., for representing details on techniques, places where the artwork was located previously, etc. It also involves defining how the user will select his preferred concern (explicitly, by indicating his interests, etc). However, there is another problem which is central to our discussion: how do we model artworks? how do we indicate which information should be shown when accessing the artwork according to different concerns? And particularly, how do we deal with this extension seamlessly regarding the previous schemata.

An elegant solution, discussed with detail in [17], is to characterize new roles for the *Artwork* node: *Technical*, and *Historical* as shown in Figure 13. These roles contain the additional information (and outgoing links) which correspond to each of these concerns. This is a simple, compact and expressive way of extending the diagram with concern information. Moreover, and similar to other examples above, we can add new concerns (i.e., roles) in a seamless way, as we don't need to modify the base Artwork class.

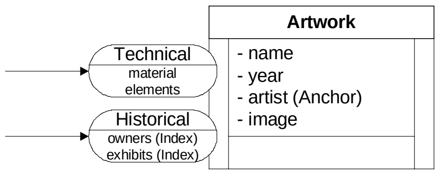


Figure 13 Improving Artworks for concern driven navigation.

An interesting additional issue arises when one analyzes the knowledge relationship between roles and the corresponding object. While in the basic model (e.g., the one in [47]) roles know the object, nothing is said about the inverse: do objects know their roles?

In most applications, this is not necessary and, besides, it compromises modularity, as the addition of a new role cannot be done transparently. However, in some cases we might need it, mainly at the conceptual level. In the example of Figure 5, when asking information about an artwork, we would want to know if it is in restoration (i.e., if it is currently playing this role). The best solution for this is to make artworks aware of the roles they are actually playing. Some comments on the solution of this problem can be read in the next sub-section.

4.4 Implementing Roles in Web Applications

Most Web Engineering approaches are implementation independent, which means that the documents generated during the modeling and design activities can be mapped to any Web software platform. Particularly, both OOHDM and UWE models can be easily implemented in current platforms like J2EE, .Net, etc. either using plain object technology or combinations between objects and relational databases. Both approaches also have associated tools which translate designs into implementations semi-automatically. Though this paper focuses on conceptual modeling of Web applications and discussing implementation issues is not its main focus, our aim in this sub-section is to show how we are implementing the role abstraction in a straightforward way by using existing languages and architectures.

There are basically two problems which must be addressed: the definition of classes in the application model and the generation of Web pages from the nodes' specification in the navigational schema. From now on, and to make the discussion concrete, we assume an object-oriented implementation, using any framework which implements the Model View Controller metaphor for Web applications [29].

The implementation of model classes supporting the role concept can be done using the Role Object pattern [4], which basically extends the Decorator pattern [15] to support the intended semantics of roles. The mapping from conceptual roles (those shown in Figure 5) to classes in the Role Object pattern is simple. Typing aspects might arise according to the programming language of choice. These issues have been extensively discussed in the object-oriented literature (See for example [15]). Additionally, the Role Object pattern implementation schema allows not only that roles know their base objects but also vice versa, thus allowing more flexibility in managing roles, without a significant loss of modularity.

The implementation of navigational roles might be more challenging as nodes are not generally defined as first-class objects in implementation settings (e.g., those using MVC), but are built dynamically (for example using Java Server Pages). In this way, we cannot map the navigation class structure directly into Java classes. Instead, a smart solution is to rely on object's builders [15]. A builder in the context of MVC is an object which is used to create the corresponding view (page) according to a controller request. In this case, we will have a hierarchy of builders for a given node class. The abstract class contains a template method [15] with the algorithm for creating the common node's structure and each sub-class contains the algorithm which corresponds to each of the possible roles. In this way, when a link is triggered (an action captured by the controller), the builder object that corresponds to the actual role of the link target is instantiated. This object executes the template in the abstract class (e.g., ArtworkBuilder) which begins the creation of the page; in turn, this method invokes the additional methods in the receiver (e.g., ArtworkInExhibition) which complete the specification. In this way, the view is completed and can be displayed.

5 Related Work and Discussion

For the sake of clarity, we structure our comparison into two sub-sections, i.e., roles in hypermedia and roles vs. other modeling mechanism, emphasizing those research works that are related with our proposal.

5.1 Roles in Hypermedia

Some authors have already proposed the use of the role abstraction in software engineering methods as explained in Section 3. However, so far, the concept has been almost ignored in the hypermedia and Web engineering community.

An interesting exception is [1]. The authors propose a role-based modeling approach in which different navigational schema are built as role models and nodes are described as different roles of conceptual classes, thus unifying the conceptual and navigational schema. The main motivation of this approach is to overcome the lack of support for logical modeling of navigation in class-based approaches (like OOHDM, WebML and UWE), and to take into account navigation issues already at the domain level. Role models can be seen as independent conceptual models and, for each context, one can have a role model. The authors claim that this additional dimension in conceptual modeling can lead to clearer and more focused models, better customized to different domain contexts [1]. The problem of this approach is twofold. On the one side, it does not visualize the relationship between roles and natural types. If the application requires the definition of several roles for different concepts, there is no possibility to recognize how they are related. On the other side, there is not a clear definition of the semantics of the modeling elements (nodes and edges) used in the visual representation they propose. There is not a clear separation of concerns followed by a separate modeling of the domain, the navigational and presentational aspects.

On the contrary, we propose to enrich both conceptual and navigational models with roles, while preserving the viewing or mapping relationship between these two models, as defined in current methods. In our proposal, materialized as explained in Section 5 with different notations according to the chosen method, roles express different sets of behavioral services for an object in the conceptual model and different “faces” of a node when being navigated in different contexts.

WSDM [11] is a method for defining audience-driven Web applications. As it is based in ORM, it “inherits” ORM’s role modeling style (described in Section 3.2). WSDM has been improved by the use of concurrent task trees [12] to model tasks. The modeling foundations of WSDM and ours are different since we focus on object-oriented models while WSDM inherits the best practices of data modeling extending ORM to model functionality. Object-oriented roles and ORM objects and roles are different and can not be compared without addressing a much deeper comparison: object-oriented modeling vs. data modeling, which is obviously outside the scope of the paper.

The navigation model in OO-H [16] is defined by means of a navigation access diagram (NAD). One of the modeling elements of this diagram is the navigation class, which is defined as a view on a conceptual class. Navigation classes are depicted with the class symbol and different views on a same conceptual class can be used in a navigation access diagram. Different roles are then defined as different navigation classes related to the same class of the domain. The idea behind this modeling technique is easy to understand and corresponds to the general idea of navigation model as a view of the conceptual model learnt from OOHDM [43]. The notation OO-H proposed for these views are instead confusing the object-oriented designer insofar as they use the notation for “object of a class” (name of the object: name of the class) to indicate “view of a class”.

Though not completely related with our research, it is worth mentioning the relationships among role-based hypermedia design and adaptive hypermedia [13]. While the former is a (static) design mechanism to improve specification and provide variability in objects´ collaboration patterns, the latter is an approach for dynamically changing content and link topologies according to the individual user’s profile. In an adaptive hypermedia application, nodes might show different information or outgoing links according to the user’s profile and preferences, his navigation history and other environmental or contextual information. Adaptive hypermedia design methods like [13] provide tools for expressing the rules that will dictate the corresponding adaptation.

Therefore, the focus in adaptive hypermedia lies in the specification of those rules and in indicating how the rules’ actions will change the appearance of nodes’ attributes and links. Additionally, rules can be applied to outgoing links to dynamically calculate the link target. Roles meanwhile, attack a much more general software engineering problem: how to indicate the way in which the sender of a message influences the behavior of the receiver (at the level of application models) and how the origin of a link influences the target of the same link (at the navigation model). Roles are a way to specify richer object classes, by using role types. Therefore, a fundamental difference between the intent of adaptive hypermedia and roles is that while the former deals with more dynamic aspects (e.g., choosing which links should be exhibited), the latter deals with a more static typing problem.

Conventional Web applications (e.g., those which do not exhibit adaptive behaviors) need, as repeatedly shown in the paper, the use of roles, while they might not need adaptation rules. In other words techniques for adaptation are not necessary in cases in which roles are. While it should be possible to use adaptation rules to provide the same effect achieved with the use of roles, being their intent much broader, we might pay two unnecessary costs: first, while roles are a type construct, which express in an encapsulated entity the intended extensions or addition to a class, rules tend to grow in flat sets and, therefore, might be more difficult to maintain. Besides, rule systems (at least those used by adaptive hypermedia) need some run time support (e.g. a rule manager or interpreter), which is not necessary if the problem can be solved just using type composition, as it is finally the case of a role-based implementation.

Meanwhile, a good adaptive hypermedia design can benefit from the use of roles. A simple case in which this can happen is when certain incoming links to a node need adaptation (i.e., it is necessary to apply the adaptation rules), while others don’t need it (the node will be shown as specified). An elegant solution we propose to the problem is to define the corresponding node as possibly playing one role: adaptive; those incoming links that need adaptation will “enter” into the adaptive role, as sketched in Figure 14. The adaptive role includes the execution of the adaptation rules.

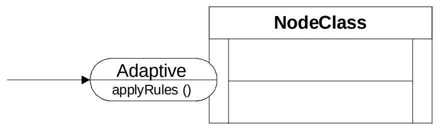


Figure 14 Using roles to indicate when adaptation rules are necessary.

This solution has two advantages; first, it allows that one specific node class could have a plain (non-adaptive) behavior and second, by decoupling the adaptation behaviors and locating it in the

Adaptive role type, it simplifies evolution, as the base node class is not polluted with the adaptation code. This solution can eventually been used to extend the scope of roles to considering the previous navigation path (instead of the link's origin). In this last case, it is difficult, if not impossible, to specify the corresponding context-aware response (where the context is the previous path) by just using the notion of roles (as a type enrichment). As the nature of paths can vary dynamically, a rather static solution like the one provided by role-based specifications does not scale, and more dynamic conditions should be specified. In this case, the role in Figure 14 could contain the code which checks the context (e.g. using rules) and decides the information and links to show.

5.2 Roles vs. Other Modeling Mechanisms

The discussion above is related with the use of design primitives and abstraction mechanisms during the Web Engineering life cycle. We think that the design concerns of conceptual (application) and navigation modeling are different in essence and thus different modeling primitives and abstraction mechanisms are necessary.

In object-oriented approaches (as the ones discussed in this paper), application modeling and design must be done using well-known object-oriented heuristics and guidelines; a clear identification of classes, their attributes and behaviors is critical during this process. The role abstraction helps to improve this activity by allowing to specify those object behaviors that depend on relationships with other objects. As we show in this paper, it is not possible to model this kind of variations using other modeling primitives, such as sub-classes as the intent of generalization/specialization hierarchies is different. To enforce the modeling power of the role construct, it is worth saying that some authors propose to use roles as a specification mechanism to solve the kind of problems which gave rise to aspect-oriented programming [50].

Meanwhile, navigational design has its roots in mature hypertext theories. Even though navigational components are specified using software engineering methods, the guidelines and heuristics for building a good linking topology are different from those aimed at obtaining a good object-oriented design. It has been shown elsewhere (See for example [44]) that a good navigational model requires opportunistic design decisions both for defining nodes' attributes, links, indexes, and other hypertext structures. Some of these decisions such as those related with the relationship among conceptual and navigational objects are better expressed using a viewing mechanism than with the role mechanism, as proposed in [1]. From a more architectural design point of view, nodes can be seen as applications of the Observer design pattern [15], while roles allow expressing different signatures (sets of services) for the same class.

As Web applications become mainstream, more complex design problems arise; it is important to use the best abstraction mechanism that is suited to solve each of those problems. Roles can be a powerful tool that must be combined with other existing (class-based) primitives and patterns. However, a deeper discussion on abstraction mechanisms is far beyond the intent of this paper though we hope to have settled the starting points for this discussion.

6 Concluding Remarks and Further work

In this paper, we have discussed the use of the role modeling abstraction in the context of the Web Engineering process. We have shown that incorporating roles in our vocabulary and design armory can help us obtain more compact and expressive design models. We have argued that introducing roles in existing Web design methods can improve their modeling power; we have also shown that existing notations for roles can, in general, be used in well-known methods maintaining their consistency. We have also discussed how our approach differs from other uses of roles in the literature.

We have demonstrated with a realistic example that using roles in the context of an object-oriented method, we can solve problems, at the modeling level, which can not be solved using existing modeling primitives and mechanisms (e.g., sub-classing).

To summarize, the use of roles in Web engineering methods allows to:

- Define key abstractions more concisely, both at the conceptual and navigational levels. We can separate the intrinsic properties of an abstraction from those which depend on the relationship (linking or behavioral) in which the abstraction is involved.
- Roles are more flexible than sub-classes, and so they can be attached to an object dynamically; Sub-classes are not a good solution for this problem.
- Roles simplify evolution; instead of modifying the core abstractions we can work with role types, for example to provide slightly different information according to a new navigation path.

We are now working on two different research areas: we are looking for good modeling practices (using the role abstraction) in order to integrate different navigational models. In most Web design methods, a different navigational model is built for each user profile (or audience); however, it is important to view these different models in a unified diagram, which might comprise repetitions. The role abstraction can be used to express this diagram more modularly.

We are also working on the use of patterns all along the engineering process of Web applications. By using a formal representation of navigation patterns with role diagrams [38], we aim to obtain a systematic approach to improve the derivation of class models from higher level (pattern-based) role models.

We strongly believe that these ideas open a wide research spectrum in the Web Engineering community. There are still many open issues such as how to effectively use roles from the early requirement stages, when roles are to be used instead of other modeling abstractions, the impact of the use of roles in mobile Web applications, etc.

Acknowledgements
We would like to thank Andreas Kraus, Daniel Schwabe, and Peter King for fruitful comments on previous versions of this paper. We also thank the rich comments of the anonymous reviewer.

References

1. Allert . H., Dolog, P., Nejdl, W., Siberski, W., and Steimann, F., Role-oriented Models for Hypermedia Construction – Conceptual Modeling for the Semantic Web -. Technical Report, Univ. Hannover (2003).

2. Aroyo, L., De Bra, P., Chepegin, V., Semantic Web-based Adaptive Hypermedia, WWW2004 Workshop Application Design, Development and Implementation Issues in the Semantic Web (May 2004).
3. Assman, U., Role-Based Design. A Concept for understanding Design Patterns and Frameworks. Available at http://www.ida.liu.se/~TDDB84/slides/7-role-based-design.pdf, (viewed Nov. 2004).
4. Bäumer, D., Riehle, D., Siberski, W., and Wulf, M., The Role Object Pattern. In Proceedings of Pattern Languages of Program Design (PloP) (1997) Available at http://jerry.cs.uiuc.edu/~plop/plop97/Proceedings/riehle.pdf
5. Bernstein, M., Patterns of Hypertext. In Proceedings of the 9th ACM International Conference on Hypertext and Hypermedia (Hypertext '98), Pittsburgh, USA (1998), 20-24.
6. Cachero, C., Koch, N, Gomez, J., and Pastor, O., Conceptual Navigation Analysis: A device and Platform Independent Navigation Specification. In Schwabe D., Pastor O., Rossi G., and Olsina L. (Eds.) Proc. of Second International Workshop on Web-Oriented Software Technology (IWWOST02) (2002) 21-32.
7. Ceri, P., Fraternali, P., and Bongio, A., Web Modeling Language (WebML), A Modeling Language for Designing Web Sites. Computer Networks and ISDN Systems, 33(1-6), June (2000) 137-157.
8. Chen, P. P., The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. Database Syst. 1(1) (1976) 9-36.
9. Dahchour, M., Pirotte, A., and Zimanyi, E., A Role Model and its Metaclass Implementation, Information Systems 29 (2004) 235–270.
10. Depke, R., Engels, G., and Küster, J. M., On the Integration of Roles in the UML. Technical Report No. 214, University of Paderborn, August 2000.
11. De Troyer, O., Audience-driven web design", In Information modeling in the new millennium, Matt Rossi & Keng Siau (Eds.), Publ. IDEA GroupPublishing (2001).
12. De Troyer, O., Casteleyn, S., Plessers, P, "Using ORM to Model Web Systems". In Proceedings of Workshop on Object-Role Modeling 2005. Springer Verlag LNCS 3763, pp 700-709, 2005.
13. Dolog, P. and Bieliko, M., Navigational Modeling in Adaptive Hypermedia. In Proc. of Int. Conference on Adaptive Hypermedia (AH 2002), Malaga Spain, May 29-31,. Paul De Bra, Peter Brusilovsky, Ricardo Conejo (Eds.). LNCS 2347, Springer Verlag (2002) 586-591.
14. Fowler, M., UML Distilled. Addison Wesley (1997).
15. Gamma, E., Helm, R., Johnson, and R., Vlissides, J., Design Patterns: Elements of reusable Object-Oriented Software. Addison Wesley, Reading (1995).
16. Gomez, J., Cachero, C., and Pastor, O., Conceptual Modeling of Device Independent Web Applications. IEEE Multimedia 8(2) (2001) 26-39.
17. Gordillo, S., Rossi, G., and Schwabe, D., Separation of Structural Concerns in Physical Hypermedia Models, In Proceedings of CAiSE 2005, LNCS, Springer (2005) 446-459.
18. Gordillo, S., Rossi, G., Laurini, R., and Schwabe, D., Decoupling Geographic from Conceptual Information in Physical Hypermedia Models. DEXA Workshops 2005, 443-447.
19. Gottlob, G., Schrefl, M., and Böckl, B., Extending Object-Oriented Systems with Roles, ACM TOIS, Vol.14 (3) (1996) 268-296.
20. Gronbaek, K., Kristensen, J., and Eriksen, M., Physical Hypermedia, Organizing Collections of Mixed Physical and Digital Material. Proceedings of the 14th. ACM International Conference of Hypertext and Hypermedia (Hypertext 2003), ACM Press, 10-19.
21. Guell, N., Vilain, P., and Schwabe, D., Modeling Interactions and Navigation in Web Applications. In Proceedings of the International Workshop on Conceptual Modelling and the Web (2000) 115-127.
22. Halpin, T., UML Data Models from an ORM Perspective: Part Five, Journal of conceptual modeling, Issue 5 (Oct. 1998).
23. Hinz, M. and Kappel, G., UML & Work (in German), dpunkt Verlag (1999).
24. Jodlowski, A., Habel, P., Jacek, P., and Subieta, K., Extending OO Metamodels towards Dynamic Object Roles. On The Move to Meaningful Internet Systems 2003, CoopIS: DOA and ODBASE, Catania, Sicily, Italy (Nov. 2003) Proceedings. LNCS 2888, Springer (2003), 1032-1047.
25. Journal of Web Engineering, Vol 12(4), Special Issue on Adaptive and Intelligent Web-based Educational (2004).
26. King, P., Nanard, M., Nanard, J., and Rossi, G., A Structural Computing Model for Dynamic Service-based Systems. Symposium MetaInformatics, LNCS 2994 (2003).

27. Koch, N., Kraus, A., and Hennicker R., The Authoring Process of UML-based Web Engineering Approach. In Proceedings of the 1st International Workshop on Web-Oriented Software Construction (IWWOST 02), Valencia, Spain (2001) 105-119.
28. Koch, N. and Kraus, A., Towards a Common Metamodel for the Development of Web Applications. In 3rd International Conference on Web Engineering (ICWE 2003) Cueva, J., Gonzalez, B., Joyanes, L., Labra, J., and Paule, M. (Eds) LNCS 2722, ©Springer Verlag (July 2003) 497-506.
29. Krasner, G., and Pope, S., A Cookbook for Using Model-View-Controller User Interface Paradigm in Smalltalk-80", Journal of Object Oriented Programming, August/September (1988) 26-49.
30. Kristensen, B.B. and Osterbye, K., Roles, Conceptual Abstraction Theory and practical Language Issues. Theory and Practice of Object Systems, 2(3) (1996) 143-160.
31. Mosse, F., Modeling Roles. A Practical Set of Analysis Patterns. Available at http://www.objectdiscovery.com/papers/roles/
32. Nanard, M., Nanard, J., and Kahn, P., Pushing Reuse in Hypermedia Applications. Golden Rules, Design Patterns and Constructive Templates. In Proceedings of the 9 th. ACM International Conference on Hypertext and Hypermedia (Hypertext '98), Pittsburgh, USA (1998) 11-20.
33. Nanard J., Nanard M., and King P., A Hypermedia-based Model for Integrating Open Services and Metadata. 14th. International ACM Conference Hypertext'2003, ACM Press (2003) 128-137.
34. Object Management Group, The UML 1.5 Specification. In http://www.omg.org/uml/
35. Pastor, O., Fons, J., and Pelechano, V., OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models, Third International Workshop on Web-Oriented Software Technologies, 2003.
36. Pernici, P., Objects with Roles. Proceedings of the ACM-IEEE Conference on Office Information Systems (1990) 205-215.
37. Reenskaug, T. M. H., Working with objects. The OOram Software Engineering Method. Manning/Prentice Hall (1996).
38. Riehle, D., Describing and Composing Patterns using Role Diagrams. In Proc. Ubilab Conference I (1996) 137-152.
39. Riehle, D. Composite Design Patterns. In Proc. OOPSLA'97 (1997) 218-228.
40. Riehle, D., Role Model Based Framework Design and Integration. In Proceedings of the 1998 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98). ACM Press (1998) 117-131.
41. Rossi, G., Schwabe, D., and Lyardet, F., Improving Web Information Systems with Navigational Patterns. Computer Networks 31, Elsevier (1999) 1667-1678.
42. Rossi, G and Schwabe, D., Abstraction and Reuse Mechanisms in Web Applications Models. In Proc.s of the International Workshop on Conceptual Modelling and the Web (2000).
43. Schwabe, D and Rossi, G., An Object-Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems (TAPOS), Vol 4 (1998) 207-225.
44. Schwabe, D. and Rossi, G., Web Application Models are more than Conceptual Models. In Proceedings of the International Workshop on Conceptual Modelling and the Web (1999).
45. Schwabe, D., Rossi, G., Conference Review System in OOHDM. In Proceedings of the International Workshop on Web Oriented Software Technology, Valencia, Spain (2001) Available at http://www.dsic.upv.es/~west2001/iwwost01/
46. Sowa, J., Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley (1984).
47. Steimann, F., On the Representation of Roles in Object-Oriented and Conceptual modeling. Data and Knowledge Engineering 35 (2000) 83-106.
48. Steimann, F., A radical revision of UML's Role Concept. In Proc. of the Unified Modeling Language Conference 2000, LNCS, Springer (2000) 194-209.
49. Steimann, F., Role=Interface: A Merger of Concepts. Journal of Object-Oriented Programming, Oct./Nov. (2001) 23-32.
50. Steinman, F., Domain Models are Aspect Free. In Proceedings of MoDELS 2005, LNCS, Springer Verlag (2005) 171-185.
51. UWA consortium, The UWA Approach to Modeling Ubiquitous Web Applications, in Proceedings of the 2002 IST Mobile & Wireless Telecommunications Summit, Thessaloniki (Greece) (2002).