Using Hadez to formally specify the Web museum of the National Gallery of Art

Daniel M. German
University of Victoria

Hadez is a formal specification language for the design of data-intensive Web applications. In this paper, we use it to describe the Web museum of the National Gallery of Art (www.nga.gov). Hadez divides the specification of a Web application into three main parts: its conceptual schema, which describes the domain-specific data and its relationships; its structural schema, which describes how this data is combined and gathered into more complex entities, called composites; and the perspective schema, which uses abstract design perspectives to indicate how these composites are mapped to hyperpages, and how the user interacts with them. Hadez provides a formal framework in which properties of a specification can be specified and answered.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications; H.5.4 [**Information Systems Applications**]: Hypermedia/Hypertext

General Terms: Design

Additional Key Words and Phrases: Hypermedia design, formal specification, Web design.

1. INTRODUCTION

Specifications have different purposes:they serve as documentation; they serve as a mechanism for generating questions; they can be used as a contract between the designer and the customer; and good specifications facilitate implementation and maintenance. Specifications can be informal and formal. Informal specifications—in particular those written in a natural language—present deficiencies that make them unsuitable for rigorous development of applications [Meyer 1985]. Formal specifications, on the other hand, use a mathematical notation to describe, precisely and unambiguously, the properties which an information system must have without unduly constraining the way in which those properties are achieved [Spivey 1992]. Questions regarding the characteristics of the final application can be answered with confidence by analyzing the design instead of analyzing the final application.

A formal specification language for hypermedia design would allow a developer to specify a hypermedia application unambiguously and to verify certain properties about it. Such a language will assist the designer to find errors in the design which otherwise can only be found during the implementation or testing of the application. Maintenance is also improved. In the first place, the specification serves as documentation, and second, the ability to verify the specification will allow the maintainer to realize the impact of changes in the design before actually implementing them. When following a formal method, the specification of a system is the most important part of its design and development [Hall 1990].

This paper introduces Hadez by showing the specification of a virtual museum (the National Gallery of Art, in Washington D.C.). It describes the characteristics of the content needed to create the site, how this content is combined to create higher level entities (called

This research was supported by BC ASI, NSERC, and the University of Victoria.
Author's address: D. German, dmgerman@uvic.ca, Department of Computer Science University of Victoria, PO Box 3055, STN CSC, Victoria BC V8W 3P6.

composites, in Hadez) and then how these composites are presented to the reader. We finish by describing the formalism behind Hadez and how it can be used to verify properties of the specification.

2. AN OVERVIEW OF HADEZ

Hadez is an object oriented specification language for hypertext design [German 2000]. Hadez is derived from the formal specification languages Z [Spivey 1992] and Z++ [Lano 1992], and further extended with constructs oriented towards the specification of hypermedia. Hadez is based on OOHDM [Schwabe and Rossi 1995] and divides the specification of an application into three parts: conceptual schema, structural schema, and perspective schema.

2.1 Conceptual Schema

The conceptual schema of Hadez is a collection of given types, type constructors, classes definitions, relations and instances. This part of Hadez is very similar to a Z structural specification (as opposed to a behavioral one). Hadez allows the declaration of classes and single inheritance.

2.2 Structural Schema

The structural schema is a collection of Hadez composite schemas. A composite schema describes how to create a composite from other composites or from data from the conceptual schema. A composite schema has three main parts: its parameters section, which specifies what composites or data are required to instantiate the composite; a sequence of free variables; and a set of predicates that bind those free variables to instances in the hyperbase or to other composites. A composite schema in Hadez looks like a Z schema, with two main differences: in Hadez the name of the composite is preceded by the Greek letter Ξ; and, composites can inherit their characteristics from other composites. This inheritance is simply a syntactic facility that allows reuse of the characteristics of other composites. Another feature of Hadez is ability to define generic composite schemas. A generic composite schema's main difference from a typical composite schema is its use of generic types. A generic type is used like a given type (that is, its details are unknown), and it must be instantiated before it can be used (using a generic type instantiation). The main objective of generic composite schemas is, like templates in C++, to avoid repetitive constructs and promote reuse of the specification.

2.3 Perspective Schema

In HDM, the term *perspective* is used to refer to different ways in which the same composites can be presented to the reader; HDM states that the specification of the perspectives is outside the scope of the methodology [Garzotto et al. 1993]. In OOHDM, the concept of perspective is served by Abstract Design Views (ADV). As HDM and OOHDM do, Hadez separates the specification of application domain objects from how they are perceived by the reader. The same object can be presented in a variety of forms, each particularly suited for the objective of the application or for a particular type of reader. In order to accomplish this task, Hadez uses the notion of Abstract Design Perspective or ADP. An ADP describes how a reader should perceive a given object and how the interaction with the user alters this perception. It specifies also what becomes an anchor of a hyperlink and what it means to select such an anchor. ADPs are based on ADVs and can be considered their descendants. An ADP observes a composite and one composite can be observed by one or more ADPs at the same time. In Hadez, an ADP models a perspective. The main characteristics of ADPs are:

—An ADP has a collection of local attributes that define a set of states. This set of states can be infinite.

—An ADP can change its state only through input messages. Input messages are generated by the reader, by the run-time system or by other regions within the current viewport (the total area visible by the reader occupied by the hypermedia system). A change from one state of the region to another is known as a transition.

—A transition can trigger an output message that is intended for ADPs in other regions.

—The ADP specifies, for each state, what parts of the observed composite are shown to the reader. The set of attributes shown to a reader in a given state is known as the *current selection* of the ADP. A transition of the ADP might determine a change in the current selection. The ADP, therefore, has the ability to present different information to a reader at different times.

—An ADP can consist of a finite number of ADPs. The communication between the different component ADPs is done through internal messages.

The viewport of a runtime system is divided into regions. Each of these regions is modeled with one ADP. The result is that the viewport corresponds to an ADP, potentially consisting of one or more ADPs. The main goals of ADPs are to be able to specify the complex interaction between the hyperbase and the reader and to promote the reuse of this specification. An ADP serves three main purposes: 1) it specifies how the composite should be broken into pages; 2) it indicates which attributes of the composite should be presented to the reader, and how they are organized within the hypernode; and 3) it specifies what attributes of the composite should be hyperlinked to other composites. A perspective is not a static entity. The user can change its state. An ADP specifies how the perspective reacts to user requests. The state of an ADP determines what parts of the composite the ADP should show at that particular state. ADPs can be composed into more sophisticated ones by using: aggregation, inheritance or parallel composition. ADPs do not replace ADVs, in fact, a specification that combines both can gain from the expressiveness qualities of each modelling artifact. The perspective schema is composed of a sequence of ADPs.

3. THE NATIONAL GALLERY OF ART WEB SITE

Museums make very good subjects for the demonstration of design methods and tools. Garzotto et al. [1995] illustrated HDM using the Microsoft *Art Gallery*; Schwabe et al. [1995] demonstrated OOHDM with a Web museum for the works of Candido Portinary. Museums are composed of well defined entities (artists, artifacts) which are interrelated by simple relationships. Nonetheless, despite their logical simplicity, these entities can be presented in a wide variety of ways, making them a good example. In this section, we demonstrate the use of Hadez by writing a specification for the Web museum of the National Gallery of Art.

The Web museum of the National Gallery of Art Web Museum, in Washington (NGA-WM—www.nga.gov) is one of the best web sites of its type [Garzotto et al. 1998]. The NGA-WM main purpose is to display the permanent and temporary collections of the museum. This museum is considered a well-designed hypermedia application.

The main sections of the museum are: *General Information*, *The Collection*, *Exhibitions*, *Online Tours*, and the *Gallery Shop*. In order to keep the size of this specification manageable we are going to specify only the collection and the online tours sections of the museum.

The main entities of the NGA-WM are artifacts, of five different types: paintings, sculptures, decorative arts, works on paper and architecture. The museum also includes information about artists. The core of the museum is a collection of virtual tours. These tours show sets of artifacts. The site includes four different types of tours: *collection tours, in-depth study tours, architecture tours*, and *virtual exhibit tours.* In order to keep the size of this specification manageable, we are not going to include architectural artifacts or tours as part of it. Figure 1 is an entity-relationship diagram showing the main classes of the museum and their relationships.

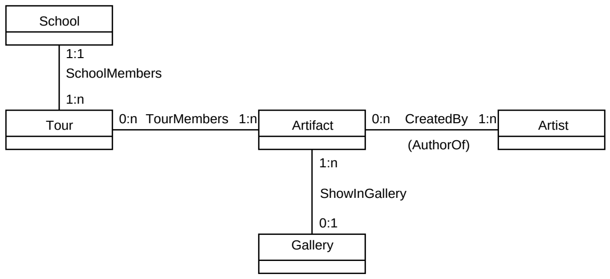

Fig. 1. Entity-relationship diagram showing the different classes in the museum application and their relationships. The museum is composed of a large group of artifacts, created by artists; the artifacts are shown in tours (virtual tours in the web site) or in galleries (which correspond to the galleries in the real-world museum). The tours are organized in schools (a school is a group of artists under a common influence).

Figure 2 shows an OOHDM diagram showing the main navigational contexts created in this application. For instance, from the main page it is possible to access a list of all the schools (a school is a group of artists under a common influence), ordered by type; or a list of all the artists, ordered by name. The artifacts are accessible from either a tour, an artist or a gallery.

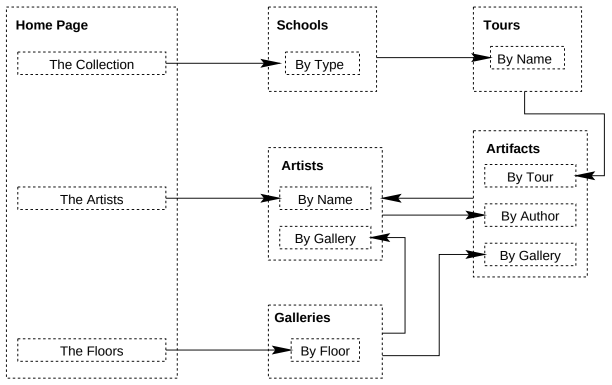

Fig. 2. OOHDM diagram depicting the navigational properties of the virtual museum.

4. COLLECTIONS

The museum's collection is composed of art pieces (artifacts). Art pieces can be categorized into four different classifications: paintings, decorative art, sculptures and work on paper. These types are represented by the enumerated type *ARTIFACT_TYPE*.

$$ARTIFACT\_TYPE = (painting, decorativeart,\\ sculpture, workonpaper)$$

Several given types will be required by the specification, namely, *TEXT*, used to represent large blobs of text; *DATE* for, as its name implies, date; *IMAGE* which represents a digital photograph; *ACCESSION_TYPE* which represents a unique identifier for an artifact within the museum; *STRING* is used to represent sequences of characters; and *FLOOR_TYPE* which represents the different floors of the two buildings of the museum.

$[TEXT, DATE, IMAGE, ACCESSION\_TYPE,$
$\quad STRING, FLOOR\_TYPE]$

4.1 The classes

The main class of the application is *artifact*. This class is composed of a series of fields that need to be displayed by the application (due to lack of space, we use ellipsis "..." when we do not include the complete definition of the schema).

Artifact
Name : *STRING*
AccessionNumber : *ACCESSION_TYPE*
TypeOfArtifact : *ARTIFACT_TYPE*
CreationDate : *DATE*
...

Artists are an important part of the application. Every artifact shows facts about its corresponding artist; furthermore, the museum shows a listing of all artists that have artifacts in the collection. Similar artifacts that do not have a known artist are assigned a "particular anonymous artist". For example, the artist *British 20th Century* gathers all artifacts from anonymous artists in Britain, created during the 20th Century in Britain. These anonymous artists are treated as any other type of artist, in which *BirthYear* and *DeathYear* are empty.

Artist
Name : *STRING*
BirthYear : *STRING*
DeathYear : *STRING*
Nationality : *STRING*
Biography : *TEXT*

Artifacts are always shown as part of a tour. Apparently, the museum only creates tours of artifacts of the same type.

Tour
Name : *STRING*
Overview : *TEXT*
TypeofTour : *ARTIFACT_TYPE*

Tours are collected into groups of similar tours, called "schools". For example, the tours *"Manet and His Influence", "Camille Pissarro, Vincent van Gogh, Paul Czanne", "Paul Gauguin", "Claude Monet" and "Edgar Degas"* all are part of the school *"French Painting of the 19th Century"*. All tours in a *School* should have the same type.

School
Name : *STRING*
Description : *TEXT*
TypeOfArtifacts : *ARTIFACT_TYPE*

The NGA-WM also displays information about the current, past and future exhibitions on display at the physical museum. These exhibits do not usually have an equivalent Web exhibition; instead, the NGA-WM displays only a textual description of the corresponding exhibitions.

Exhibition

$$\begin{array}{l} Name : STRING \\ StartingDate : DATE \\ EndingDate : DATE \\ Description : TEXT \end{array}$$

The museum presents maps of the physical building, allowing the reader to view what is in each of its galleries.

Gallery

$$\begin{array}{l} Name : STRING \\ Floor : FLOOR\_TYPE \\ FloorPlan : IMAGE \end{array}$$

The museum shows special *virtual exhibitions* on a specialized topic. These exhibitions are different to tours, and they appear to be designed each one at a time. For the sake of space, this specification will not include these exhibitions.

Finally, we define some instances of the previously declared types:

$Collection : \mathbb{P}\, Artifact$	[The entire collection of artifacts]
$AllArtists : \mathbb{P}\, Artists$	[All the artists displayed in the collection]
$AllTours : \mathbb{P}\, Tour$	[All the electronic tours of the NGA-WM]
$TourOfTheWeek : Tour$	[A weekly tour, selected from AllTours]
$AllSchools : \mathbb{P}\, School$	[All the schools]
$AllGalleries : \mathbb{P}\, Gallery$	[The set of all galleries]
$AllFloors : \mathbb{P}\, FLOOR\_TYPE$	[The set of all floors]

4.2 The relationships

One of the most important relationships is *CreatedBy*, which relates artists with artifacts. Every artifact in the collection has at least one creator, and similarly, each artist might create one or more artifacts. The domain of the relationship is the collection, and its range is the set of artists represented in the museum.

$$\begin{array}{l} CreatedBy : Artifact \leftrightarrow Artist \\ \hline \operatorname{dom} CreatedBy = Collection \\ \operatorname{ran} CreatedBy = AllArtists \end{array}$$

AuthorOf is the inverse of *CreatedBy* and will be useful to find the artifacts created by a particular artist.

$$\begin{array}{l} AuthorOf : Artist \leftrightarrow Artifact \\ \hline AuthorOf = CreatedBy \sim \end{array}$$

Tours are composed of a non-empty sequence of artifacts. All the artifacts must be of the same type and they must match the type of the tour. The function *TourMembers* maps a given tour to its corresponding sequence of artifacts.

$$
\begin{array}{|l}
\mathit{TourMembers} : \mathit{Tour} \rightarrow \mathrm{seq}_1\, \mathit{Artifact} \\
\hline
\forall\, T : \mathit{Tour} \bullet \\
\quad \mathit{TourMembers}(T) \bullet \\
\quad\quad \forall\, i : 1..\#\mathit{TourMembers}(T) \bullet \\
\quad\quad\quad \mathit{TourMembers}(T)(i).\mathit{TypeOfArtifact} = \\
\quad\quad\quad\quad T.\mathit{TypeOfArtifact}
\end{array}
$$

In a similar way, *SchoolMembers* is a function that maps a given *School* to its component set of *Tour*. Because the component tours do not have to be ordered, the functions return a non-empty set of *Tour*.

$$
\begin{array}{|l}
\mathit{SchoolMembers} : \mathit{School} \rightarrow \mathbb{P}_1\, \mathit{Tour} \\
\hline
\forall\, \mathit{TG} : \mathit{School};\ \mathit{TSeq} : \mathbb{P}_1\, \mathit{Tour} \bullet \\
\quad \mathit{SchoolMembers}(\mathit{TG}) = \mathit{Tseq} \Rightarrow \\
\quad\quad \forall\, i : 1..\#\mathit{Tseq} \bullet \mathit{Tseq}(i).\mathit{TypeOfTour} = \\
\quad\quad\quad \mathit{TG}.\mathit{TypeOfArtifacts}
\end{array}
$$

Location is a function that maps artifacts to the gallery in which they are located (in the physical galleries of the museum).

$$
\begin{array}{|l}
\mathit{Location} : \mathit{Artifact} \mathrel{+\!\!\!\rightarrow} \mathit{Gallery} \\
\hline
\mathrm{dom}\, \mathit{ShownInGallery} \subseteq \mathit{AllArtifacts} \\
\mathrm{ran}\, \mathit{ShownInGallery} = \mathit{AllGalleries}
\end{array}
$$

ShownInGallery is a function that returns the artifacts being displayed in a given gallery. Because these are the physical galleries, its predicate section states that no artifact can be shown in two different galleries.

$$
\begin{array}{|l}
\mathit{ShownInGallery} : \mathit{Gallery} \mathrel{\rightarrowtail\!\!\!\!\!+} \mathbb{P}_1\, \mathit{Artifact} \\
\hline
\mathit{ShownInGallery} = \mathit{Location}\, \sim
\end{array}
$$

GalleriesInFloor returns the set of galleries that exist on a given floor. It is defined as a injective function and therefore it is not necessary to specify that each gallery can only appear in one floor.

$$
\begin{array}{|l}
\mathit{GalleriesInFloor} : \mathit{Floor} \rightarrowtail \mathbb{P}_1\, \mathit{Gallery} \\
\hline
\mathrm{ran}\, \mathit{GalleriesInFloor} = \mathit{AllGalleries} \\
\forall\, F_1, F_2 : \mathit{Floor} \bullet \\
\quad F_1 \neq F_2 \Rightarrow \mathit{GalleriesInFloor}(F_1) \cap \\
\quad\quad \mathit{GalleriesInFloor}(F_1) = \varnothing
\end{array}
$$

4.3 Creating the composites

We start by defining *ArtistComposite*, which relates artists to their creations:

$$
\begin{array}{|l}
\hline
\Xi\mathit{ArtistComposite}(A : \mathit{Artist}) \\
\mathit{Artifacts} : \mathbb{P}\, \mathit{Artifact} \\
\hline
\mathit{Artifacts} = \mathit{AuthorOf}(\!|\ \{A\}\ |\!) \\
\hline
\end{array}
$$

One of the most important composites corresponds to an artifact. *ArtifactComposite* gathers a given artifact, and its non-empty set of artists.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Xi \mathit{ArtifactComposite}(\mathit{Art} : \mathit{Artifact})} \\
\hline
\mathit{Authors} : \mathbb{P}_1\ \mathit{ArtistComposite} \\
\hline
\mathit{Authors} = \Xi \mathit{ArtistComposite} \circ \mathit{CreatedBy}(\!|\ \{\mathit{Art}\}\ |\!) \\
\hline
\end{array}
$$

A *TourComposite* gathers all the artifacts that are part of that tour (a non-empty sequence of artifacts).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Xi \mathit{TourComposite}(T : \mathit{tour})} \\
\hline
\mathit{Elements} : \mathrm{seq}_1\ \mathit{ArtifactComposite} \\
\hline
\mathit{Elements} = \Xi \mathit{ArtifactComposite} \circ \mathit{TourMembers}(T) \\
\hline
\end{array}
$$

Similarly, *SchoolComposite* gathers a set of the tours.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Xi \mathit{SchoolComposite}(S : \mathit{School})} \\
\hline
\mathit{ToursComp} : \mathbb{P}_1\ \mathit{TourComposite} \\
\hline
\mathit{ToursComp} = \Xi \mathit{TourComposite} \circ \mathit{SchoolMembers}(S) \\
\hline
\end{array}
$$

The composite *AllToursComposite* gathers all the *Tours* into a single composite.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Xi \mathit{AllToursComposite}} \\
\hline
\mathit{Tours} : \mathbb{P}\ \mathit{TourComposite} \\
\hline
\mathit{Tours} = \Xi \mathit{TourComposite} \circ \mathit{AllTours} \\
\hline
\end{array}
$$

Similarly, the composite *AllSchoolsComposite* gathers all the *Schools* into a single composite.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Xi \mathit{AllSchoolsComposite}} \\
\hline
\mathit{Schools} : \mathbb{P}\ \mathit{SchoolComposite} \\
\hline
\mathit{Schools} = \Xi \mathit{SchoolComposite} \circ \mathit{AllSchools} \\
\hline
\end{array}
$$

The *IndexOfArtists* is a set of all the artists which have at least one creation in the collection.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Xi \mathit{IndexOfArtists}} \\
\hline
\mathit{ArtistsComposites} : \mathbb{P}\ \mathit{ArtistComposite} \\
\hline
\exists\, \mathit{ArtistsWithWorks} : \mathbb{P}_1\ \mathit{Artist} \bullet \\
\qquad \forall\, A : \mathit{Artist} \bullet \mathit{AuthorOf}(\!|\ \{A\}\ |\!) \neq \varnothing \Rightarrow \\
\qquad\qquad A \in \mathit{ArtistsWithWorks} \\
\qquad \mathit{ArtistsComposites} = \Xi \mathit{ArtistComposite} \circ \\
\qquad\qquad \mathit{ArtistsWithWorks} \\
\hline
\end{array}
$$

GalleryComposite corresponds to a gallery that displays a set of artifacts:

$$\begin{array}{l}
\Xi GalleryComposite(G : Gallery) \\
\hline
DisplayedArtifacts : \mathbb{P}\, ArtifactComposite \\
DisplayedAuthors : \mathbb{P}\, ArtistComposite \\
\hline
DisplayedArtifacts = \Xi ArtifactComposite \circ \\
\qquad ShownInGallery(G) \\
DisplayedAuthors = \{Author : ArtistComposite \mid \\
\qquad (\exists\, A : ArtifactComposite \bullet A \in DisplayedArtifacts \\
\qquad \land Author \in A.Art.Authors)\}
\end{array}$$

FloorComposite will include all the galleries in a given floor:

$$\begin{array}{l}
\Xi FloorComposite(Floor : FLOOR\_TYPE) \\
\hline
Galleries : \mathbb{P}_1\, GalleryComposite \\
\hline
Galleries = \Xi GalleryComposite \circ GalleriesInFloor(Floor)
\end{array}$$

And finally, *AllFloorsComposite* gathers all the floors in the museum.

$$\begin{array}{l}
\Xi AllFloorsComposite \\
\hline
Floors : \mathbb{P}_1\, FloorComposite \\
\hline
Floors = \Xi FloorComposite \circ AllFloors
\end{array}$$

The *CollectionComposite* is composed of the tour of the week plus a set of all the tours in the virtual museum.

$$\begin{array}{l}
\Xi CollectionComposite \\
\hline
WeekTourComposite : TourComposite \\
AllSchools : AllSchoolsComposite \\
\hline
WeekTourComposite = \Xi TourComposite(TourOfTheWeek)
\end{array}$$

The *MainPageComposite* consists of: an artifact, selected at random from the collection (as a result, the image in the main page of the museum changes randomly); the entire collection of schools; the floors of the building; and index of artists.

$$\begin{array}{l}
\Xi MainPageComposite \\
\hline
RandomArtifact : ArtifactComposite \\
TheCollection : CollectionComposite \\
TheFloors : AllFloorsComposite \\
TheArtists : IndexOfArtists \\
\hline
\exists\, Random \in Collection \bullet \\
\qquad RandomArtifact = \Xi ArtifactComposite(Random)
\end{array}$$

5. DESCRIBING THE PERSPECTIVES

The presentation of composites is done with abstract design perspectives (ADPs). We first declare two perspectives, which will be defined later. By declaring them here, an ADP can refer to one of these ADPs, allowing circular references. For example, an *ArtifactADP* needs to link to *ArtifactPhotoADP*

$$\begin{array}{l}
ArtifactADP : ADP(ArtifactComposite) \\
ArtistADP : ADP(ArtistComposite)
\end{array}$$

5.1 The artifact's perspectives

Artifacts are shown in two different ADPs: *ArtifactPhotoADP*, which shows a large photo, intended to fill most of the screen, along with the basic information about the artifact; and *ArtifactADP*, a detailed page that displays all the information known about the artifact.

Figure 3 shows *ArtifactPhotoADP*. It displays a photograph of the painting followed by some of the attributes of the artifact, including the name of each of its authors.

```
ADP ArtifactPhotoADP Observes A : ArtifactComposite
        Invariants:
            A.Art.Photograph
        Block GeneralInfo:
            foreach Auth : Author in A.Authors do
                Auth.Name
            A.Art.Name
            A.Art.Owner
            A.Art.CreationDate
            A.Art.AccessionNUmber
            anchor “Information” linkto ArtifactADP(A)
End ArtifactPhotoADP
```

Fig. 3. ArtifactPhotoADP is intended to display a large photo and basic information about an artifact.

Figure 4 corresponds to the main view of an artifact. In the declarations and preconditions of the ADP, the variable *Tours* is defined. *Tours* will contain the set of tours in which the artifact is displayed. In the block *ToursOfWhichItIsPart*, the ADP will display the name of each one of those tours in which this artifact is presented. The icon in the artifact links to the *ArtifactPhotoADP* which observes the same *ArtifactComposite*. In the block *AuthorsInfo*, the ADP displays information related to each one of the authors of the painting, including a link to their corresponding *ArtistADP*.

The NGA-WM uses buttons to hide and show information from a given page. For instance, the ADP shows only one of the following attributes at a given time: artifacts description, bibliography, exhibition history, conservation notes, and provenance; button—each corresponding to each attribute—selects which one should be the only one displayed.

5.2 The tour's perspective

We now turn our attention to the description of tours. As it was previously described, a *Tour* is composed of a non-empty sequence of *Artifact*. A tour is presented to the reader in blocks of six artifacts at a time, called a “room”. If the tour has more than six artifacts, then it is divided in more than one room. In a *Tour*, the upper part of the screen will be devoted to a room, and this is going to be modeled with the ADP *ShowRoomADP*, which is defined in figure 5. This ADP will show an icon of each one of the artifacts linking to their corresponding ADP.

The *TourADP*, depicted in figure 6, is responsible for showing a *Tour*. It embeds a *ShowRoomADP* in the block *CurrentRoom* and, by using the reserved word **attach**, dynamically changes the subsequence that the current room displays. The variable *CurrentOffset* determines the index to the first element of the room (a room is always of size 6) and the function *subsequence* is used to extract, from a sequence, a certain number of elements starting at a given offset.

```
**ADP** ArtifactADP **Observes** $A : ArtifactComposite$
    **Declarations:**
        $Tours : \mathbb{P}\, Tour$
        $show\_desc$ : **internal message**
        ...
    **Preconditions:**
        $Tours = \{t : Tour \mid A.Art \in \text{ran}(TourMembers(t))\}$
    **Block** *GeneralInfo*:
        **anchor** $A.Art.Icon$ **linkto** $ArtifactPhotoADP(A)$ $A.Art.Name$
        $A.Art.PhysicalDescription$
        $A.Art.Owner$
        $A.Art.CreationDate$
        **anchor** "Full screen image" **linkto** $ArtifactPhotoADP(A)$ ...
    **Block** *Buttons*:
        **button** $desc \rightarrow show\_desc$
        ...
    **Preconditions:**
        $hide(Bibliography)$
        ...
    **Message** *show_desc*:
        $show(Description)$
        $hide(Bibliography)$
        ...
**End** ArtifactADP
```

Fig. 4. ADP for Artifacts.

```
**ADP** ShowRoomADP **Observes** $Works : \text{seq}_1\ ArtifactComposite$
    **Preconditions:**
        $\# Works \leq 6$
    **TheWorks:**
        **foreach** $A : ArtifactComposite$ **in** $Works$ **do**
            **anchor** $A.icon$ **linkto** $ArtifactADP(A)$
**End** ShowRoomADP
```

Fig. 5. The ADP responsible for showing the current room.

The buttons *start_tour*, *next_room*, and *previous_room* are used to change the value of *CurrentOffset*. When the reader presses the button *next_room*, *CurrentOffset* is incremented by 6 (the size of a room); a new subsequence of artifacts, starting in the new value of *CurrentOffset* is attached to the ADP *ShowRoomADP*.

The ADP also shows the entire list of names of artifacts and links to their corresponding ADP.

5.3 Perspectives for schools and the collection

In the ADP for a school, there is a random icon from one of the artifacts in one of the tours in the school; this icon links to its corresponding *ArtifactADP*. In the next block, there is a description of the school. Finally, it lists each of the name of the tours in the school with a link to their corresponding *TourADP*. Figure 7 shows this ADP.

The list of all schools is classified by the type of artifacts that the school displays. As a consequence, it is necessary to select—within a specific school—those groups of artifacts that are of a particular type. The function *SchoolOfType* takes two parameters: a set of

```
**ADP** TourADP **Observes** $Tour : TourComposite$
    **Declarations:**
        $CurrentOffset : \mathbb{N}$
        $show\_next\_room : internalmessage$
        $show\_previous\_room : internalmessage$
        $restart\_tour : internalmessage$
    **Block** $Title$**:**
        $Tour.Name$
    **Block** $Buttons$**:**
        **button** $start\_tour \rightarrow restart\_tour$
        **button** $next\_room \rightarrow show\_next\_room$
        **button** $previous\_room \rightarrow show\_prev\_room$
    **Block** $CurrentRoom$**:**
        **ADP** $ShowRoomADP$
    **Block** $AllWorks$**:**
        **foreach** $A : ArtifactComposite$ **in** $Tour.Elements$ **do**
            **anchor** $A.Art.Name$ **linkto** $ArtifactADP(A)$
    **Preconditions:**
        $\#Tour.Elements \geq 1$
        $CurrentOffset = 1$
        **attach** $subsequence(Tour.Elements, 1, 6)$ **to** $CurrentRoom$
        $hide(previous\_room)$
    **Message** $next\_room$**:**
        $\#Tour.Elements \geq CurrentOffset + 6 \Rightarrow$
            $CurrentOffset' = CurrentOffset + 6$
            $show(previous\_room)$
            $\#Tour.Elements < CurrentOffset' + 6 \Rightarrow hide(next\_room)$
            **attach** $subsequence(Tour.Elements, CurrentOffset', 6)$ **to** $CurrentRoom$

    **Message** $prev\_room$**:**
        ...
    **Message** $restart\_tour$**:**
        ..
**End** TourADP
```

Fig. 6. The ADP for a Tour.

```
**ADP** SchoolADP **Observes** $Sch : SchoolComposite$
    **Declarations:**
        $RandomArtifact : ArtifactComposite$
    **Preconditions:**
        $\exists\, T : TourComposite \bullet$
            $T \in Sch.ToursComposites$
            $RandomArtifact \in \operatorname{ran} T.Elements$
    **Block** $Image$**:**
        **anchor** $RandomArtifact.Art.Icon$ **linkto** $ArtifactADP(RandomArtifact)$
    **Block** $Description$**:**
        $Sch.S.Description$
    **Block** $Tours$**:**
        **foreach** $T : TourComposite$ **in** $T \in Sch.S.ToursComp$ **do**
            **anchor** $T.T.Name$ **linkto** $TourADP(T)$
**End** SchoolADP
```

Fig. 7. *SchoolADP* is responsible for displaying a group of tours.

SchoolComposite and a variable of *ARTIFACT_TYPE* and returns only those elements of the set of the specified type.

Each type of school is presented by the ADP *ListOfSchoolsADP* which observes a set of *SchoolComposite*, all of them of the same type. It first shows the type of the artifacts and then proceeds to list the name of each school, linking to the corresponding ADP.

```
ADP ListOfSchoolsADP Observes $Sch : \mathbb{P}\,SchoolComposite$
        Block $ArtifactType$:
              $Sch.S.TypeOfArtifacts$
        Block $GroupOfTours$:
              foreach $S : SchoolComposite$ in $SchoolsComps$ do
                  anchor $Sch.S.Name$ linkto $SchoolADP(S)$
End ListOfSchoolsADP
```

Fig. 8. ***ListOfSchoolsADP*** **lists the name of schools and links to their corresponding ADP.**

ListOfSchoolsADP is meant to be included into the ADP *CollectionADP*. *CollectionADP*, defined in figure 5.3, will list all the different types of artifacts and for each, select those schools that include artifacts of that type, and display them using the ADP *ListOfSchoolsADP*. The ADP also shows the name of the “Tour of the Week”, which links to its corresponding *TourADP*.

An interesting feature of *CollectionADP* is that it shows the icons of 4 random artifacts in the collection, linking to their corresponding *ArtifactADP*. These random artifacts are chosen using the function *RandomArtifacts*, which chooses 4 random artifacts from a set of *Tour*.

```
ADP CollectionADP Observes $Coll : AllCollectionComposite$
        Block $RandomArtifacts$:
              foreach $A : ArtifactComposite$ in $RandomArtifacts(Coll.AllTours)$ do
                  anchor $A.Art.Icon$ linksto $ArtifactComposite(A)$
        Block $TourOfTheWeek$:
              anchor $Coll.WeekTourComposite.T.Name$ linkto
                  $TourADP(Coll.WeekTourComposite)$
        Block $Collection$:
              foreach $T : ARTIFACT\_TYPE$ do
                  $T$
                  ADP   $ListOfSchoolsADP(SchoolsOfType($
                      $Coll.AllSchools, T))$
End CollectionADP
```

Fig. 9. ***CollectionADP*** **shows all the groups of tours in the collection.**

6. EXTENDING THE SPECIFICATION

In order to illustrate the use of inheritance, we present in this section an hypothetical extension of the museum specification that exploits the object oriented features of Hadez.

Let us assume that the museum differentiates between different types of objects. For example, paintings and photographs, to name a few. The painting adds two attributes, *Technique* and *Material*, while photographs have three more: *Film*, *ExposureData* and *PrintingData*. We can declare these classes as subclasses of *Artifact*:

Painting : *Artifact*
Technique : *STRING*
Material : *STRING*

Photo : *Artifact*
Film : *STRING*
ExpositionData : *TEXT*
PrintingData : *TEXT*

Both *Painting* and *Photo* can be used anywhere an *Artifact* is used (they are descendents of it). Therefore, a tour can also include photographs, or paintings; the tour, however will perceive them as artifacts and their extra attributes will not be visible to it.

In order to exploit the extra attributes of the new class we need to extend the *Artifact-Composite* and the *Artifact* related ADPs in order to handle the classes.

For example, to present *Photo* it is necessary to extend *ArtifactComposite* by declaring *PhotoComposite* as its child; in this particular case, the new composite does not require any further attributes, and hence, its body is empty.

$\Xi PhotoComposite(Art : Photo) : ArtifactComposite$

We can then enhance the ADP *ArtifactPhotoADP* by adding a new block that shows the photo related information.

```
ADP PhotoPhotoADP:ArtifactPhotoADP Observes A : PhotoComposite
        Block PhotoInfo:
                A.Art.Film
                A.Art.ExpositionData
                A.Art.PrintingData
End PhotoPhotoADP:ArtifactPhotoADP
```

Fig. 10. ***PhotoPhotoADP* extends the functionality of *ArtifactPhotoADP*.**

The complete specification for the virtual museum can be found in [German 2000].

7. VERIFICATION

Formal specification languages offer several advantages over informal specification languages, such as: 1) there is evidence that the use of a specification language with well defined syntax and semantics forces the designer to be more careful in the description of a system, hence increases the quality of the design. As Bertrand Meyer stated "formal notations naturally lead the specifier to raise some questions that might have remained unasked, and thus unanswered, in an informal approach" [Meyer 1985]. 2) Some properties of the specification can be verified. The basic idea behind verification is that the specification can be translated into a set of logic statements that can be used to prove or disprove properties about the specification. We are interested in answering questions about the characteristics of the hypertext application described by the design, and more particularly, characteristics of the potential browsing sessions that readers could encounter. An Hadez specification describes two main types of facts of an application: its structural characteristics, and its behavior. The structural facts describe how the composites and, in essence, what data is included in the application. The behavior of the application relates to the way that the application reacts to the reader input. Hadez provides a formal framework in which it is possible to verify the some properties such as: Is the application realizable? Is the specification type consistent? How does the specification behave?

From the point of view of Hadez, a hypermedia application is a collection of perspectives that observe a set of composites. The perspectives interact with the reader and this interaction determines how the composites are presented. From the point of view of the reader, the hypermedia is a collection of nodes which are connected through hyperlinks. From the point of Hadez, however, each node is a perspective, which in turn can be composed of one or more perspectives. The reader interacts with the perspective, by either changing its state (by pressing buttons) or by following hyperlinks, replacing the currently displayed perspective with a new one. In order to be able to analyze how perspectives interact with a reader, it is necessary to provide a model that can represent their behavior. The model we use is I/O automata [Lynch and Tuttle 1987; 1988]. Informally, an I/O automaton is a possibly infinite state automaton. Its reacts to its environment by accepting input messages. An input message forces the I/O automaton to change its state and generate an output message in response. We are particularly interested in finding an equivalent I/O automaton for each perspective; we refer to it as the characteristic I/O automaton of a perspective. The state of the automaton is defined by the internal value of the attributes of the perspective and by which attributes of its observed composite are visible. The state of the perspective, as well as the state of the automaton, can only change by reacting to messages. The perspective's input and output messages (follow hyperlink, pressed button, display anchor, etc.) correspond to input and output messages of its corresponding I/O automaton. For an I/O automaton, an alternating sequence of messages and states is known as an execution fragment. An execution sequence that starts in a starting state is known as an execution of the automaton. The problem of analyzing how a perspective behaves under the reader requests becomes a problem of analyzing the potential executions of its characteristic I/O automaton. We can therefore ask questions about the behavior of the perspective and translate them into predicates involving its characteristic I/O automaton. For example *ArtifactADP* (defined in figure 4, page 11) observes the composite *ArtifactComposite* (defined on page 7). The question "Does the *ArtifactADP* perspective always show the name of the artifact?" can be translated to "for any state in any execution of the characteristic I/O automaton of an *ArtifactADP* perspective, is the title of the artifact visible?", or its negative counterpart "is it true that there is no state in the execution of the characteristic I/O automaton of an *ArtifactADP* perspective such that the title of the artifact is not visible?". In other words, the executions of the automaton are equivalent to the behavior of the perspective and it is desirable to be able to find, for any question about the the behavior of a perspective, an equivalent predicate on the execution of its characteristic I/O automaton. There are two ways in which a property can be verified: manually and automatically—using a theorem prover. We use HTL* to specify the behavioral properties of a perspective and the complete application. HTL* is a first order temporal logic developed by Stotts and Furata [1998]. HTL* contains path quantifiers to express properties that hold over any potential browsing session. We have adapted the semantics of an atomic predicate HTL* to Hadez. The question "Does the *ArtifactADP* perspective always show the name of the artifact?", for example, can then be rewritten as $\overrightarrow{\forall}\,\Box A.Art.Name$, where the variable $A.Art.Name$ corresponds to the name of the artifact in the scope of *ArtifactADP*.

7.1 Manual verification

In order to illustrate how properties can be verified, we will proceed to prove two properties:

Lemma $(\overrightarrow{\forall}\,\Box A.Art.Name = \mathbf{t})$ for *ArtifactADP*

A quick look a the specification of *ArtifactADP* shows that *Art*.*Name* is part of the block *GeneralInfo*. The semantics of Hadez indicate that an attribute which is part of a block is always visible as long as the block is visible. There is no predicate that hides *GeneralInfo* in the message handler section of the predicate. Therefore *Art*.*Name* remains visible as long as the perspective is visible. $\Box$

Lemma $(\overrightarrow{\forall} \Diamond A.Art.Provenance = \mathbf{t})$ for *ArtifactADP*

In this case, the question is: can the *provenance* of an artifact be visible for any browsing session during the life of an artifact ADP? This proof is a little more complex. The only place in which *Art*.*Provenance* appears is in the block *Provenance*. This block is hidden at the starting state of the application (by using the predicate $hide(Provenance)$ in the preconditions. Therefore, the predicate $\Diamond A.Art.Provenance$ is equivalent to the predicate: “is the block *Provenance* eventually shown?”. The block *Provenance* is shown inside the message handler for *show_prov*. Therefore, we need to find a state in which a message *show_prov* is generated. *show_prov* can only be generated by the button *prov* which is part of the block *Buttons*. The block *Buttons* is always visible, hence the button *prov* is always available to the reader. Since we assume that the perspective is treated fairly, at some point the reader presses the button *prov* which triggers the message *show_prov*, which in turns shows the block *Provenance*, which contains the attribute *A*.*Art*.*Provenance*. $\Box$

As the complexity of specification and the predicate to prove increases, it becomes more and more difficult to provide proofs like these. Furthermore, the potential for error in the proofs also increases.

7.2 Automatic verification

The formals behind Hadez can be used to mechanize the verification of properties. In particular, I/O automata have been modeled in Isabelle/HOL [Paulson 1994; Nipkow and Slind 1995b; 1995a]. Z specifications have been translated into HOL and then verified [Bowen and Gordon 1994]. A Hadez specification can be translated into a HOL equivalent. HTL* predicates would then be translated into HOL predicates that can be checked against the specification.

8. FUTURE WORK

Formal specifications are perceived as being too difficult for a typical Web designer. It is unfeasible to expect Web designers to learn Z and higher-order logic in order to design typical Web applications. Our goal is to use the models and formalisms behind Hadez to develop a framework and a set of tools that can assist Web designers in the creation of complex, data intensive Web applications. We are currently investigating the use of a graphical notation that can be supported by Hadez. Applications could be automatically verified (such as completeness and type consistency). These diagrams could also be used to generate to prototype and to instantiate the actual application.

9. CONCLUSIONS

This article uses Hadez, a formal language for the specification of large Web applications, to describe the virtual museum of the National Gallery Museum. The specification starts by describing the underlying classes of the museum and relations that correlate them. These classes and relations are used to create higher level entities Finally, the abstract design perspectives, or ADPs, describe how the composites are to be presented to the reader. They describe which attributes of a composite are presented to the reader, and how they are linked to other APDs. They also serve as a user interfaces, in which their state determines how the composite is presented to the reader.

The data model of Hadez divides a hypermedia application in three main components: data (conceptual schema), structure (structural schema) and navigation/presentation (perspective schema). Hadez includes a framework for stating properties of the specification and for the verification of these properties. This provides assurance to the designer that the specification complies with certain user requirements. In particular, we argue how the behavior of perspectives can be modeled and analyzed. Hadez would be particularly useful in the specification of data-intensive Web applications.

REFERENCES

BOWEN, J. P. AND GORDON, M. J. C. 1994. Z and HOL. In *Z User Workshop, Cambridge 1994*, J. P. Bowen and J. A. Hall, Eds. Workshops in Computing. Springer-Verlag, 141–167.

GARZOTTO, F., MAINETTI, L., AND PAOLINI, P. 1995. Hypermedia Design, Analysis, and Evaluation Issues. *Communications of the ACM 38,* 8 (Aug.), 74–86.

GARZOTTO, F., MATERA, M., AND PAOLINI, P. 1998. To Use or Not to Use? Evaluating Usability of Museum Web Sites. In *MW'98 Museums and the Web*.

GARZOTTO, F., PAOLINI, P., AND SCHWABE, D. 1993. HDM – A model-based approach to hypertext application design. *ACM Transactions on Information Systems 11,* 1, 1–26.

GERMAN, D. 2000. Hadez, a framework for the specification and verification of hypermedia applications. Ph.D. thesis, University of Waterloo.

HALL, J. A. 1990. Seven myths of formal methods. *IEEE Software 7,* 5 (Sept.), 11–19.

LANO, K. C. 1992. Z++. In *Object Orientation in Z,* S. Stepney, R. Barden, and D. Cooper, Eds. Workshops in Computing. Springer-Verlag, Cambridge CB2 1LQ, UK, 106–112.

LYNCH, N. A. AND TUTTLE, M. R. 1987. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing,* F. B. Schneider, Ed. ACM Press, Vancouver, BC, Canada, 137–151.

LYNCH, N. A. AND TUTTLE, M. R. 1988. An introduction to Input/Output Automata. Technical Memo MIT/LCS/TM-373, Massachusetts Institute of Technology, Laboratory for Computer Science. Nov.

MEYER, B. 1985. On formal specifications. *IEEE Software,* 7–26.

NIPKOW, T. AND SLIND, K. 1995a. I/O Automata in Isabelle/HOL. In *Proc. of Types for Proofs and Programs, LNCS 996,* P. Dybjer, Ed.

NIPKOW, T. AND SLIND, K. 1995b. I/O automata in Isabelle/HOL. *Lecture Notes in Computer Science 996,* 101–119.

PAULSON, L. C. 1994. Isabelle: A generic theorem prover. *Lecture Notes in Computer Science 828,* xvii + 321.

SCHWABE, D. AND ROSSI, G. 1995. The Object-Oriented Hypermedia Design Model. *Communications of the ACM 38,* 8 (Aug.), 45–46.

SCHWABE, D., ROSSI, G., AND BARBOSA, S. 1995. Systematic Hypermedia Application Design with OOHDM. Tech. Rep. 30, Departamento de Informática, Pontifcia Universidade Católica, Rio de Janeiro.

SPIVEY, J. M. 1992. *The Z Notation: A Reference Manual*, 2nd ed. Prentice Hall International Series in Computer Science.

STOTTS, P. D. AND FURATA, R. 1998. Hyperdocuments as Automata: Verification of Trace-Based Browsing Properties by Model Checking. *ACM Transactions of Information Systems 16,* 1, 1–30.