Advanced Web Application Modeling and Design

Dr Gustavo Rossi
LIFIA. Facultad de Informática-UNLP

Laboratorio de Investigación y Formación en Informática Avanzada
Facultad de Informática, Universidad Nacional de La Plata
Calle 50 esq. 115 - Primer piso (1900) La Plata, Argentina
TE: (0221) 422 8252 http://www-lifia.info.unlp.edu.ar

Message of the Course. Web Engineering has a dark side

- Web Applications require good modeling practices (e.g. Object-Oriented)
- Modeling practices are not enough; we also need to know good design practices
- Even those applications without complex behaviors pose design challenges to the developer

Agenda and Outline

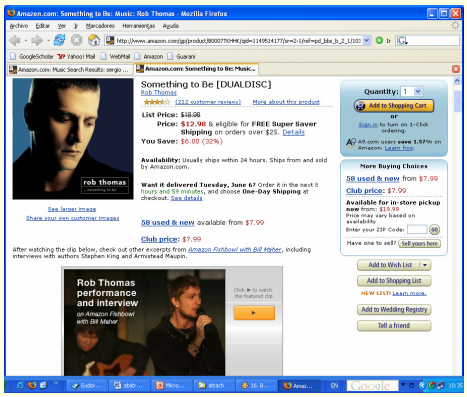
- Motivation and Examples
- Review of Advanced Separation of Concerns
- Managing Volatile Functionality
- Introducing Roles in Web Applications
- Business Processes in Web Applications
- Personalizing Web Applications
- Separation of Concerns in Requirements

Motivation and Examples

- Complex Web Software deal with a multitude of design concerns
- They offer varied and complex functionality
- Even the well-known applications (not just the research prototypes) show these aspects

Let us explore Amazon.com

The Amazon example

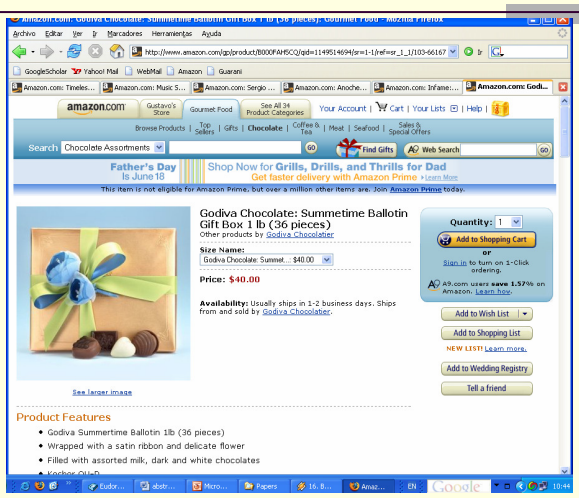


Notice the different kind of functionality; some is stable, other volatile

Another example

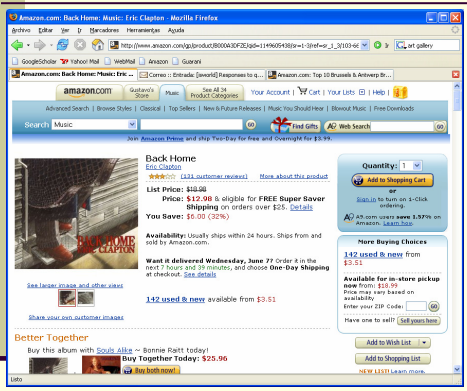
The Emi draw of tickets for the Rolling Stones concert

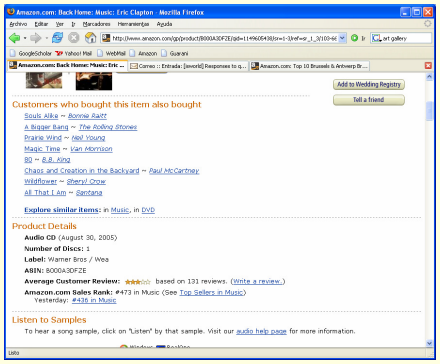
Products support different behaviors



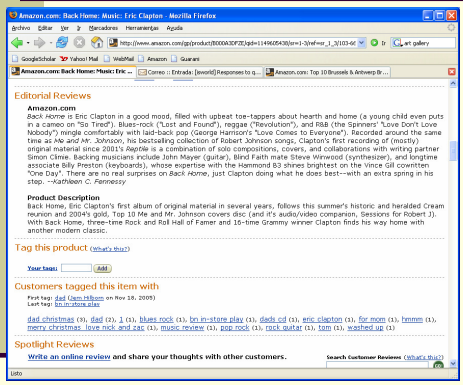
Can we sell our chocolates as we sell our CDs or books?

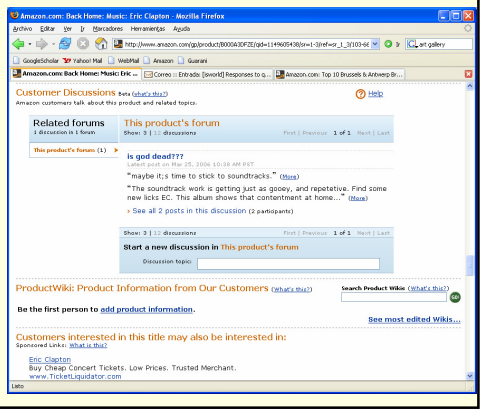
Product pages comprise multiple concerns



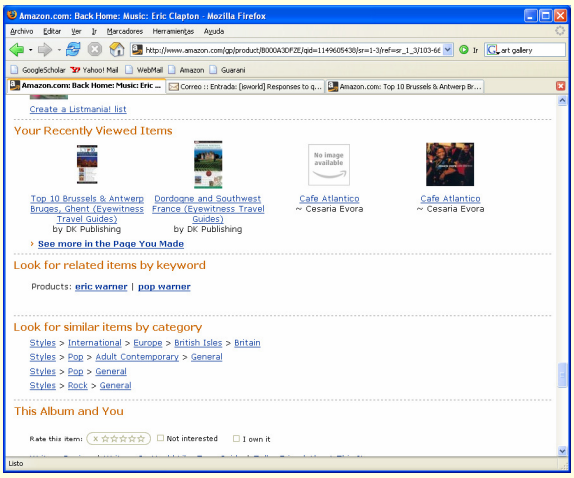


And…


Amazon.com: Back Home: Music: Eric Clapton - Mozilla Firefox
Editorial Reviews
Amazon.com
Product Description
Tag this product
Customers tagged this item with
Spotlight Reviews
Write an online review and share your thoughts with other customers.

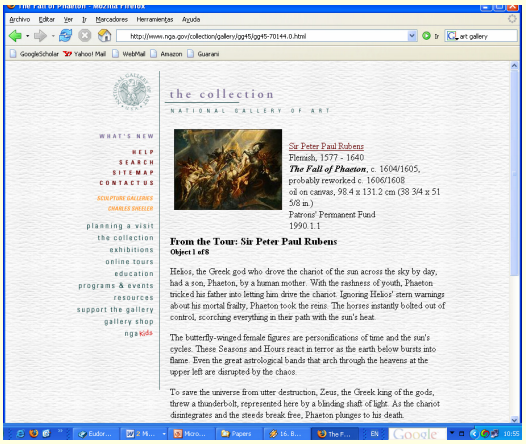

Amazon.com: Back Home: Music: Eric Clapton - Mozilla Firefox
Customer Discussions
Related forums
This product's forum
is god dead???
ProductWiki: Product Information from Our Customers
Be the first person to add product information.
Customers interested in this title may also be interested in:
Eric Clapton
Buy Cheap Concert Tickets. Low Prices. Trusted Merchant.
www.TicketLiquidator.com

And…


Amazon.com: Back Home: Music: Eric Clapton - Mozilla Firefox
Create a Listmania! list
Your Recently Viewed Items
Top 10 Brussels & Antwerp Bruges, Ghent (Eyewitness Travel Guides)
by DK Publishing
Dordogne and Southwest France (Eyewitness Travel Guides)
by DK Publishing
Cafe Atlantico
~ Cesaria Evora
Cafe Atlantico
~ Cesaria Evora
See more in the Page You Made
Look for related items by keyword
Products: eric warner | pop warner
Look for similar items by category
Styles > International > Europe > British Isles > Britain
Styles > Pop > Adult Contemporary > General
Styles > Pop > General
Styles > Rock > General
This Album and You
Rate this item:
Not interested
I own it

Finally…

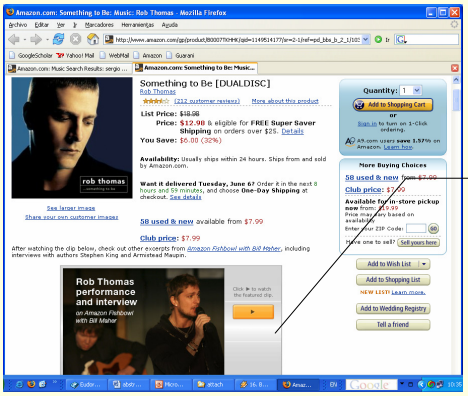
- When we navigate to an object, the context is relevant (e.g. “a la” OOHDM)



Which are our software building tools?

- We will base our discussions on the use of “advanced” approaches for separation of concerns
- Fundamental for our discussion are:
  - Design Patterns
  - Other techniques for separation of concerns, such as aspects

Volatile Functionality



How does we solve this situation?

Is this a navigational or conceptual problem?

Exercise

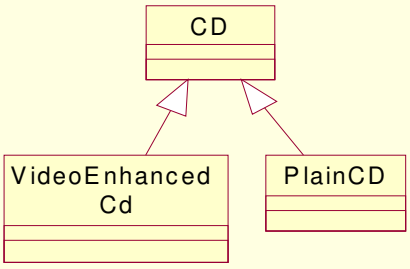
- Model the CD class and its more important relationships in UML
- Build the navigational Diagram corresponding to this part of the site
- Discuss possibilities with respect to the volatile functionality

15/20 MINUTES

Possible Solutions

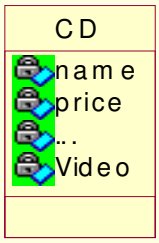
- In the conceptual model:
  - A Class Hierarchy
  - An attribute in Class CD
  - A relationship with the FishBowl

Possible Solutions



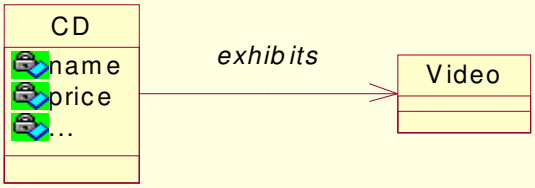
Problems with this approach?

Possible Solutions



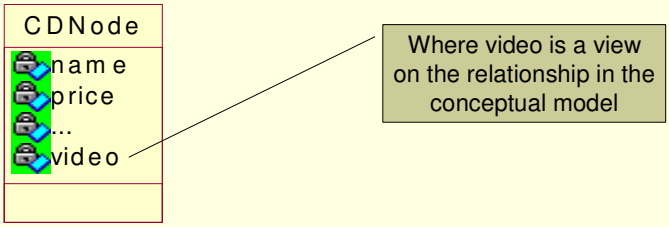
Problems with this one?

Possible Solutions



In the Conceptual Model

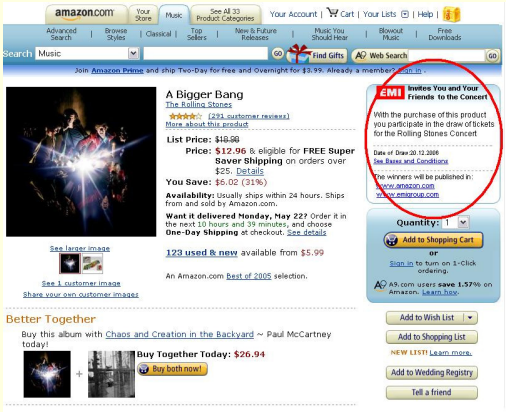
Possible Solutions



In the Navigational Model

Why these approaches fail?

Another Example



Which are the differences in this example?

Slight Changes in Behaviors

We can not sell our Chocolate

Is this a peculiarity of a sub-class of Product?
What if so? What if not?

Why volatile functionality is a problem?

- Should we clutter design models? (edit and then edit again?)
- Should we manage the problem at implementation time? (Why not?)

An Abstract Point of View

- We need to enrich the behavior of some objects
- This enrichment is dynamic (in run-time)
- The pattern of enrichment is irregular (not all instances of a class)

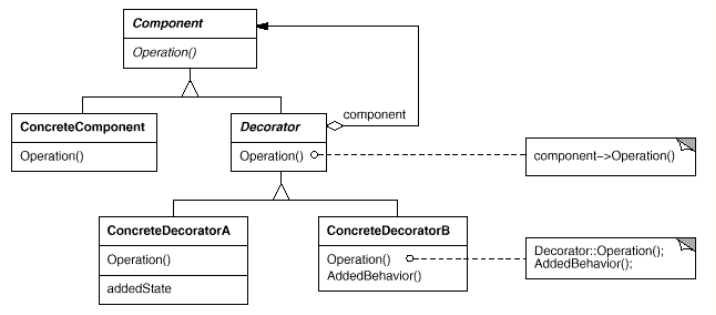
Decorators to the Rescue

Pattern Decorator

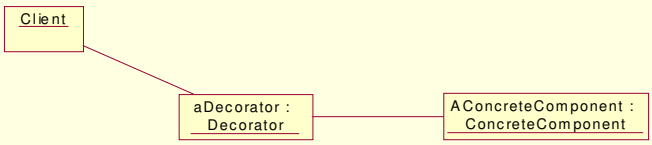
- **Intent:**

  Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality

Structure of a Decoration



Anatomy of Decoration



Clients Interact with Decorators instead of Components

Exercise

- How do we solve the problem of the CD enhanced with Video?
- Which are the trade/offs in design?
- What we gain? What we loose?
- Decorators in Conceptual Models vs. Navigational Model

15 Minutes

Implementing Volatile Models

- Strategy:

  Viewing Volatile Functionality as a model decoration

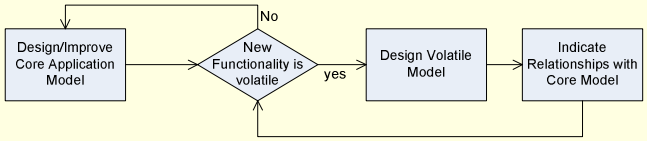
- However:

  How to deal with irregular enhancements in the navigational model

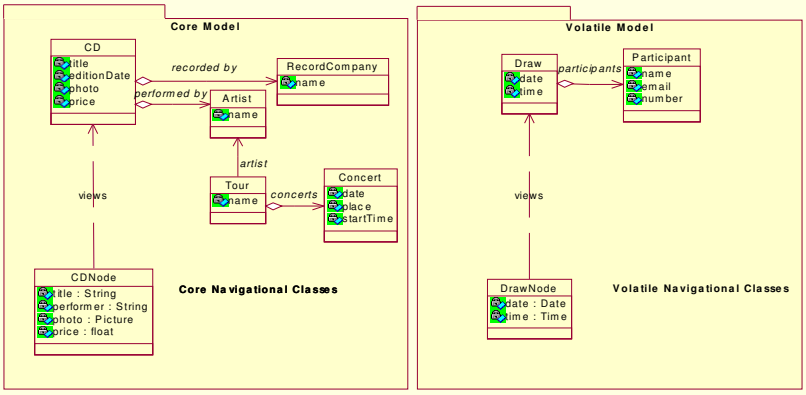
An Extension of OOHDM for volatile functionality

- We decouple volatile from core functionality: We define two design models; a core model and a model for volatile features (called VService Layer).
- New behaviors, i.e. those which belong to the volatile functionality layer are modeled as first class objects, e.g. following the Command [4] pattern.
- To achieve obliviousness, we use inversion of control, i.e. instead of making core classes aware of their new features, we invert the knowledge relationship. New behaviors know the base classes on top of which they are built.
- We use a separate integration layer to bind core and volatile functionality. In this way, we achieve reusability of core and volatile features and manage irregular extensions.

Volatile Functionality in OOHDM

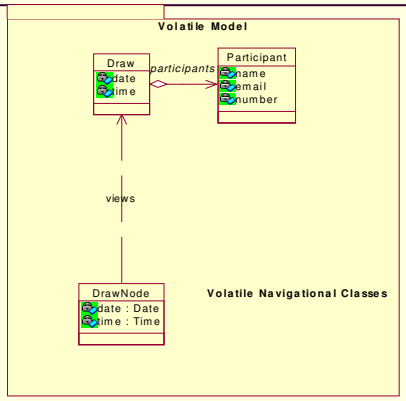


Example: Draw for tickets



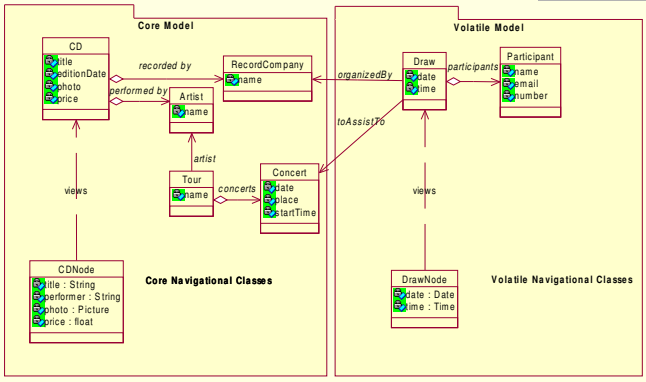
1-Separate core functionality from volatile functionality

Example…



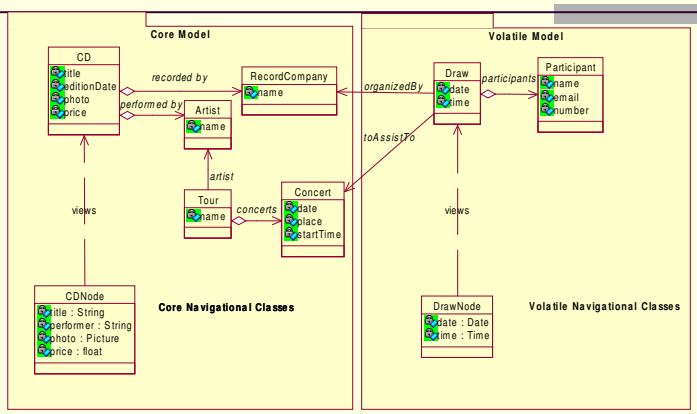
2- Behaviors, which belong to the volatile functionality layer are modeled as first class objects, following the Command pattern.

Example….



3-To achieve obliviousness, we use inversion of control, i.e. instead of making core classes aware of their new features, we invert the knowledge relationship. New behaviors know the base classes on top of which they are built.

Our Solution



4- We use a separate integration layer to bind core and volatile functionality, in order to achieve reusability of core and volatile features and manage irregular extensions.

The Affinity of a Volatile Service

- It is defined as the set of nodes from which the service can be accessed
- We use a specification in the style of the OOHDM node definition language

FROM $C_{1...Ci}$ *WHERE predicate*

Ci indicates the classes to which the WHERE predicate is applied

Examples of Affinities

Affinity Draw
From CDNode *where* (performer = Rolling Stones)
Integration: Linkage (DrawNode)
Additions: [DrawSpec: Text.
Conditions: Anchor (ToConditions)
Results: Anchor (ToResults)]

Instance Affinity RobThomas (Fishbowl)
FROM CDNode WHERE title = 'Something to Be'
AND performer = 'Rob Thomas'
Integration: Extension

Examples of Affinities

Instance Affinity StephenKingFishbowl (Fishbowl)
FROM BookNode WHERE author = 'Stephen King'
AND bookTitle = 'Cell'
Integration: Extension

What other kinds of integrations can we make?

Advantages of the Approach

- By decoupling the integration from the services and the core model, we can use different integration schemas at different time and for different instances. For example, with new CDs it is reasonable to extend the CD with the live performance; we can later use a Linkage relationship and later eliminate the access to the service.
- Using different affinity specification we can fine tune the way in which different objects are affected by the service.
- Obliviousness allows that the evolution of core and service classes is independent from each other and also from the integration specification.

Exercise

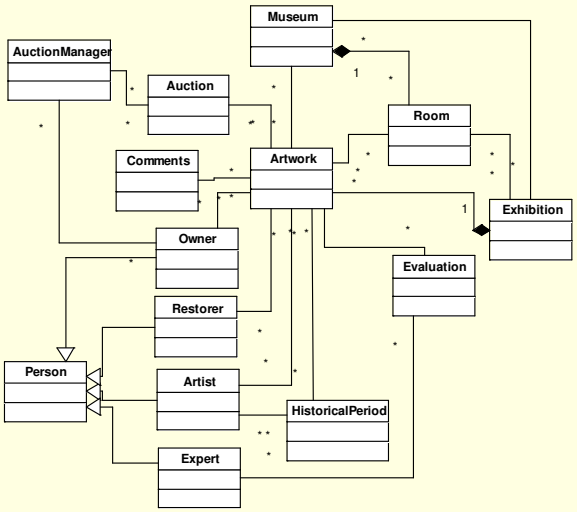
- Given the previous approach for volatile services, devise a possible implementation stategy thinking in terms of well-known web architectures

15 Minutes

Exercise

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)

Simplified Conceptual Model



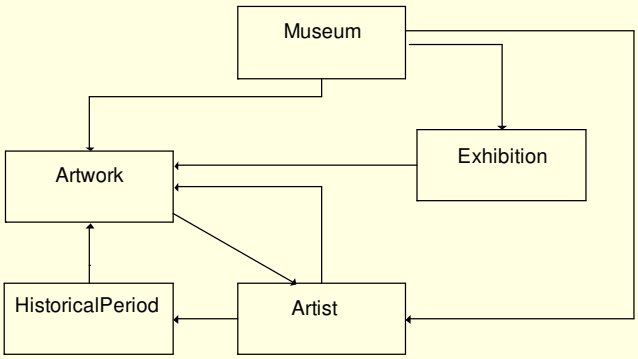
Problems with this model

- Artwork exhibits too many unrelated services:
  - For example, when being schedule for an artwork, it has to “know” the auction date and details.The actual price, doesn’t make sense in other contexts
  - Auctions in Restoration also need specific information.
- What information is “intrinsic” of an Artwork? what information depends on a relationship?

Problems with this model

- The Person class hierarchy seems correct but:
  - What if the same person is an owner and an artist
- What information is intrinsic of a Person?

In the Navigational Model



Which similar problems arise?

Example

From Artist

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

From Exhibition

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 Ch'lteau of Cloux , near Amboise in France , where he died in 1519.

Summary of Problems: During Conceptual Modeling

- How to separate behavioral concerns which depend on relationships to simplify class' interfaces?
- How to indicate those aspects (attributes and behaviors) which are only meaningful in a concern?

Summary of Problems: During Navigation Modeling

- How to indicate that the same node shows different information according to the incoming link?
- How to indicate that outgoing links also depend on the access path?

Natural Types vs Role Types

- Natural types are those types such that an instance of the type can not end belonging to the type without loosing its identity; Natural type properties does not depend on any collaboration
- Examples: Person, Artwork

Role Types

- A role type 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. A role type characterizes the dynamic state of an entity when it is involved in some collaboration with others
- Examples: Artist, Owner, InAuction

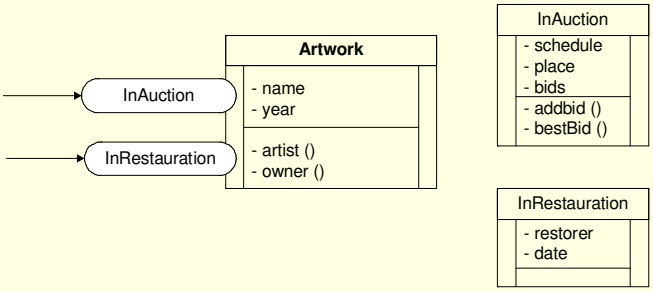
Objects play Roles

- 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*.

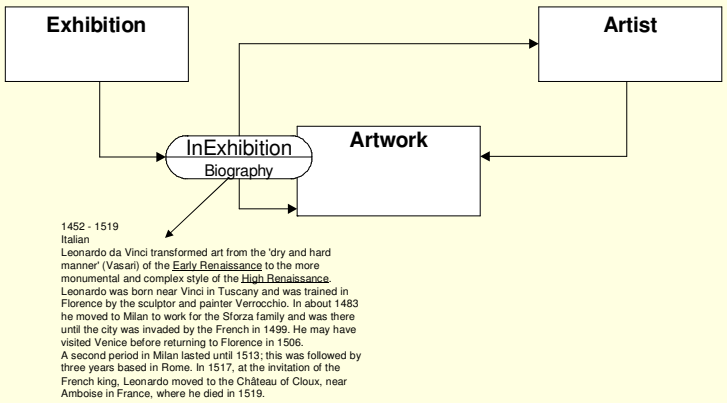
Further Concepts

- Role Types vs Role Instances
- Role Types as Classes
- Role Generalization/Specialization

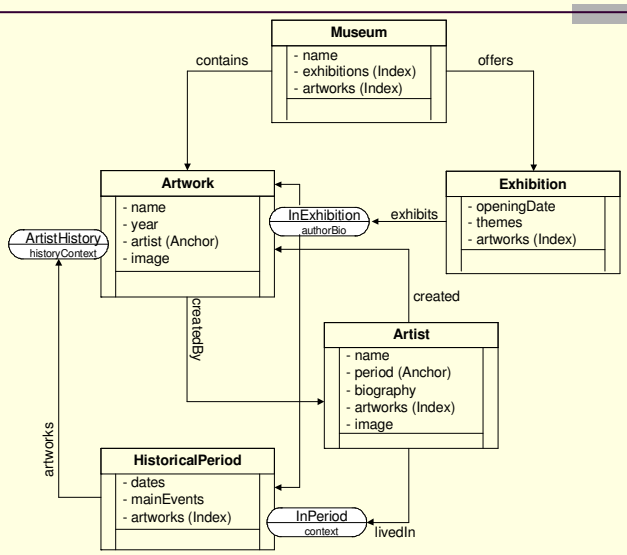
Roles in the Conceptual Model



Roles in the Navigational Model



A more complete model



Discussion

- Implementing Roles
- Constraints on Roles
- Roles vs. Decorators (are they the same)