Engineering Web Applications: Challenges and Perspectives

Daniel Schwabe
TecWeb Laboratory
Departamento de Informática,
PUC-Rio

TecWeb

Summary

- Introduction
- Challenges
- Design Issues for Web Applications
- Useful Abstractions
- Lessons Learned
- Conclusions

Summary

- Introduction
- Challenges
- Design Issues for Web Applications
- Useful Abstractions
- Lessons Learned
- Conclusions

Introduction

- First web applications - browsing documents
  - static pages
- Functionality beyond navigation added
  - dynamic pages
  - business logic
- Multiple platforms and devices – mobile/ubiquitous web
- Semantic Web

My Own Bias

- **Web applications are "Advanced Information Systems"**
- **Solution to complex problems is done by a *man-machine team*; the part done by the machine is an Advanced Information System**
- **Advanced Information Systems allow knowledge representation:**
  - "informally", when processed by the human being (hypertext/hypermedia)
  - "formally", when processed by the computer (AI, KBSs, DBs, IR, etc...)
  - Boundary between formal and informal is arbitrary and can be moved

My Own Bias

- **Hypertext paradigm is used to**
  - Help humans process informally represented knowledge
  - Integrate both representation
- **Interactivity**
  - Paradigm shift - non-sequential, user-controled
  - Time dependent data

Summary

- Introduction
- Challenges
- Design Issues for Web Applications
- Useful Abstractions
- Lessons Learned
- Conclusions

Challenges

- **Increased client expectations**
  - faster turnaround
  - requirements constantly changing
  - continuous development!
- **Principled development process**
  - allow maintenance and evolution
  - allow reuse of known solutions
- **End user development**
  - “one shot” applications
  - usually non-programmers

Challenges

- **Multi platform – hardware and software**
  - desktop PC's, Laptops, PDA's, Cell Phones, embeded devices, etc...
  - in spite of standards, browsers don't always agree in implementation
- **Support multi-disciplinary teams**
  - graphics designers
  - content producers (text, audio, video, ...)
  - marketing
  - etc...
- **Adaptable to user and environment**

Summary

- **Introduction**
- **Challenges**
- **Design Issues for Web Applications**
- **Useful Abstractions**
- **Lessons Learned**
- **Conclusions**

Design Issues for Web applications

- How do we characterize what tasks are to be supported?
- What are the information items?
- How does one navigate and process information items?
- How are information items perceived?
- How do we take the user into account in the application itself?
- Can we reuse designs effectively?
- Can we be systematic in the process?

Some premises

- Should be Model-based
  - allow abstractions to control complexity
- No single model solves it all!
- Should support various possible software architectures
- Should have a diagrammatic notation whenever possible
- Domain Specific Languages (DSLs) should be employed when possible

Graphical Notations

- Are graphical notations really easier?
- Human being has special purpose hardware – cognitive apparatus
- Map visual properties onto domain properties
  - shape
  - color
  - position
  - size
- Be consistent in the mapping
- Adequate choice of visual property still an art...
- Can't express everything graphically!

Some premises

- A good Web application is a good hypermedia application
- We will use OOHDM/SHDM as a reference
  - http://www.oohdm.inf.puc-rio.br:8668

Summary

- Introduction
- Challenges
- Design Issues for Web Applications
- Useful Abstractions
- Lessons Learned
- Conclusions

Useful abstractions

- User Interaction Diagrams
- Conceptual Model
- Navigation Model
- Abstract Interface
- Domain Specific Languages (DSLs)
- Design Patterns
- Frameworks
- Design Rationale

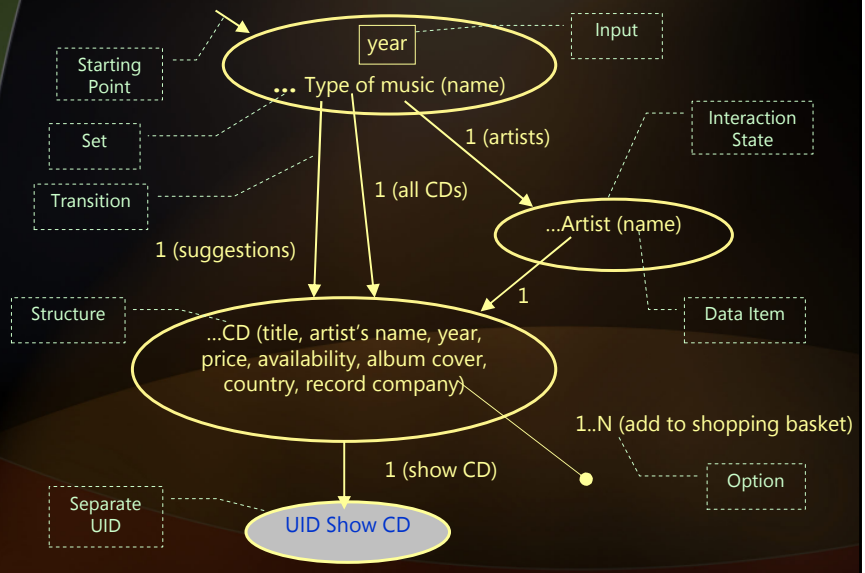
Characterizing Tasks

- **Several design methods employ Scenarios and Use Cases**
- **Web applications allow the user to navigate through information items using their navigational structure.**
- **Business logic is separate from navigation**

Characterizing Tasks

- **User Interaction Diagrams (UIDs)**
  - diagrammatic modeling technique
  - focus exclusively on the information exchange between the application and the user.
  - UIDs consider neither user interface aspects nor navigation aspects.
- **UIDs support the synthesis of**
  - conceptual model
  - navigation structure
  - interface elements

An Example of UID



Navigation Design

- What are the navigation objects?

NewsStory
title: string
content: text
photo: image*
publicationDate: date

?

Navigation Design (II)

- Navigation objects are *views* over conceptual objects!

NewsStory {from m: NewsStory}
title: string author: a: Author, <a.name *where* a IsAuthorOf m> author-pic: a:Author, <a.photo *where* a IsAuthorOf m> content: text photo: image* publicationDate: date

Navigation Design – Topology of the Navigation Space

- Node and link descriptions are too low level
- We need some abstraction analogous to the "Class" abstraction with respect to objects
- Describe sets of objects that behave analogously wrt navigation

Navigation Design - Context

- **We can define Navigation Contexts as sets of objects that have similar navigation properties**
- **Every navigation object is always accessed within a context**

$$\frac{\text{Navigation Object}}{\text{Context}} \equiv \frac{\text{Object}}{\text{Class}}$$

Defining Navigation Contexts

- **A context is characterized by**
  - A query for selecting its elements
  - An ordering for accessing its elements
  - An optional parameter
- **Depending on the query they can be**
  - class based (a filter on class attributes)
  - relation based (derived from an 1-n relation)
  - both

Navigation Contexts - Class Derived

- **Simple class based – Filter elements of a class :**
  - "CDs whose genre is *Samba*".
  - Context = {e | P(e), e ∈ C}
- **Class based group – Parameterized set of simple contexts**
  - "CDs by Genre"
  - Group = {$Context_{genre}$}, $Context_{genre}$ = {c | c.genre=*genre*, c ∈ CD}.

Navigation Context - Link Derived

- **Link based – based on an 1-to-n relationship.**
  - "All CDs by Tom Jobim"
  - Context = {p | "Tom Jobim" IsAuthorOf p, p ∈ CD}.
  - Structural links are a particular case

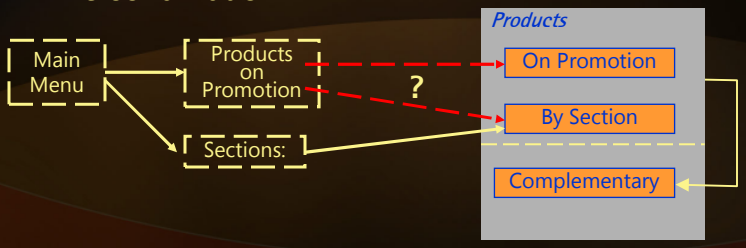
Navigation Context - Link Derived

- **Link based group – Based on an 1-n relationship where the source instance can vary**
  - "CDs by Author"
  - Group = {AuthorContext},
    AuthorContext =
    {c | a IsAuthorOf c, p ∈ CD, a ∈ Person}

Example of Context

Abstraction: Expressing Marketing Requirements

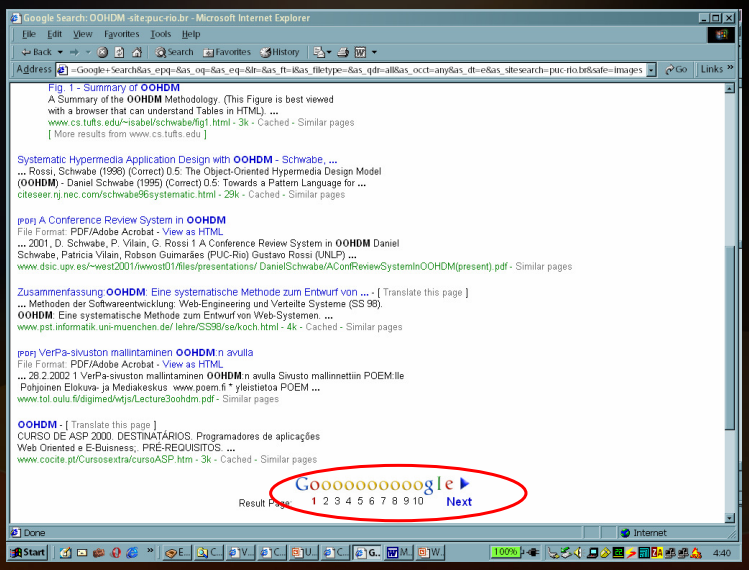
- **How should navigation be for items on promotion in an e-store?**
  - "Cross-sell"
  - "Loss Leaders" e "Up-sell"
  - Promotions
  - Personalization



Interface Design

- **Interface design is decoupled from navigation**
  - Not everything you click is a link!
- **Abstract Interface**
  - Interface is composed of a set of perceivable interface objects
  - Presentation objects are mapped to navigation objects
  - Events in the interface trigger either navigation or functions (business logic)
  - Perceivable objects change as a result of event processing
- **Allows independence from technology, standards, devices**

Interface Operations



Proposed Approach

- **Factor the interface specification in two level – abstract interface and concrete interface**
- **Abstract Widget Ontology**
  - Describes interfaces focusing on information exchange aspects
- **Concrete Widget Ontology**
  - Describes concrete interface widgets commonly found in implementation environments
- **Ontologies currently use OWL**

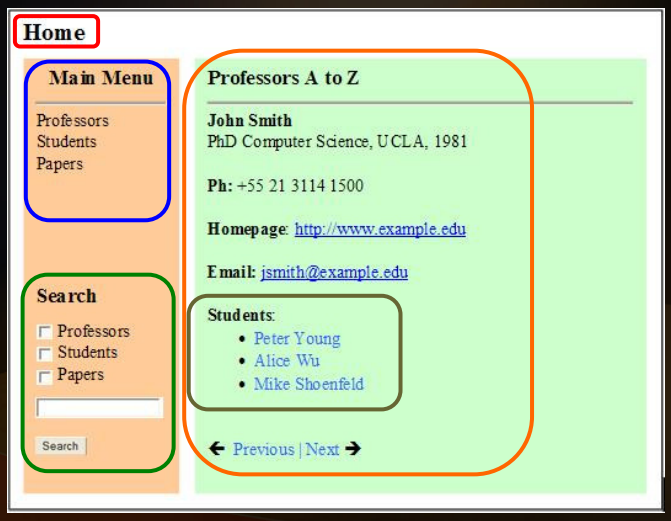
Abstract Widget Ontology

- **ElementExihibitor – exihibits some kind of content**
  - Label
  - Text
  - Image
- **SimpleActivator – reacts to external events**
  - Anchor
  - Button

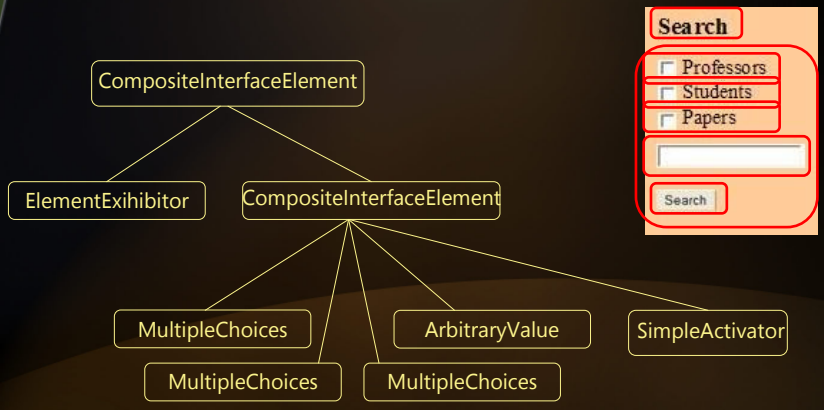
Abstract Widget Ontology

- **Capturer/ArbitraryValue – is able to capture some arbitrary input value**
  - Single-line text box
  - Multi-line text box
- **Capturer/PredefinedOptions – the value captured is chosen from a given set**
  - Radion button
  - Check box
  - Combo box

Concrete Interface Example



"Search" Component



"Professor Data" Element

DSL - Simplifying code

- Domain Specific Languages allow direct manipulation of modeled objects
- Can be achieved by dynamically extending existing programming language
- Overload primitive programming language with new, domain-related semantics

Ruby / HyperDE-DSL

Native Classes

Methods for persistence:
find, find_all, find_by_*
create, save, destroy

```
schwabe = Professor.find_by_name
                        "Daniel Schwabe"
hypermedia = ResearchArea.find_by_name
                                "Hypermedia"

schwabe.advises.each do |student|
  unless
  student.works_in.include?(hypermedia)
    student.works_in << hypermedia
  end
end
```

Link access Methods

Link access Methods

Link value assignment methods

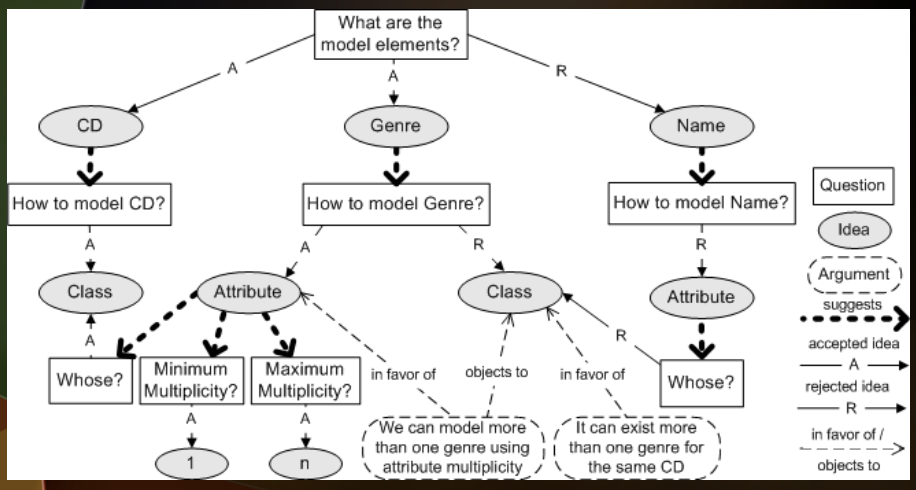
Reuse

- **"In the small" reuse of components**
  - code fragments
  - html fragments
- **Micro-architectures**
  - Design Patterns
- **Full Architectures**
  - Frameworks
- **Design Rationale**
  - Design decision structure

Design Rationale – Kuaba Ontology

- **Vocabulary to represent design decision structure**
  - Artifact
  - Idea
  - Argument
  - Decision
  - ...
- **Support both reuse and group design**
- **Assumes designed artifact is described in a formal model**

Kuaba - Example



Navigation Patterns

- **Help to record and convey good and recurrent navigation architectures**
- **Can be organized in catalogues and used as "books of experience"**
- **Examples:**
  - Landmark (to access all important sub-sites)
  - News (to indicate new products)
  - Portal (to serve as a gateway to a set of services)
  - Set-Based Navigation

Domain-specific patterns

- **In some domains, it is possible to find regular structures of problem-solution pairs**
- **Example: In e-commerce,**
  - Opportunistic Linking (for keeping the user engaged)
  - Advising (for helping the user find products he may like)
  - Explicit Process (for helping the user understand application workflows)
  - Secure Bactrack (for maintaining consistency in navigation operations)

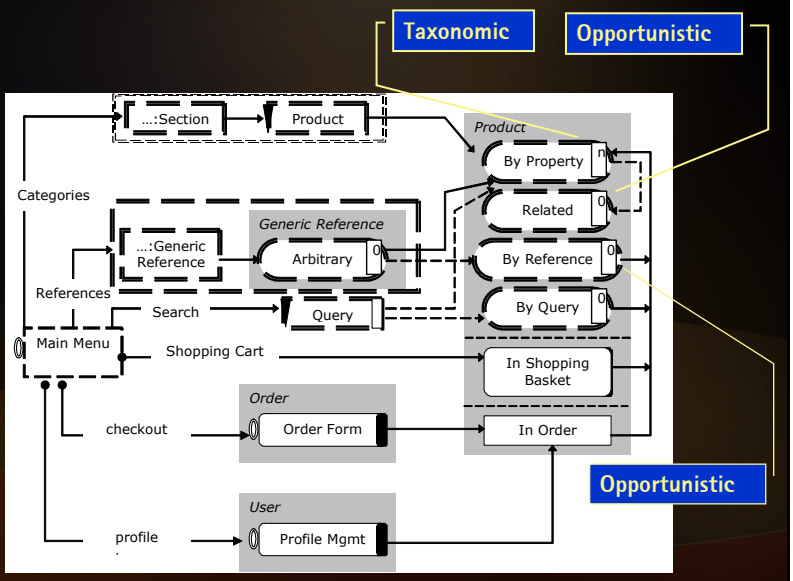
Web Frameworks

- **Frameworks are skeletons of applications in a domain**
- **Extending the notion of framework to the Web domain:**
  - Genericity in the conceptual model
  - Genericity in the navigational model (generic nodes and contexts)

OOHDM-Frame

- **Uses OOHDM models and notations as a basis for defining frameworks**
- **A Framework is defined by a set of schemas, containing "hot spots", and instantiation rules**
  - Conceptual Class Schema
  - Navigation and InContext Class Schema
  - Context Diagram and Context Cards
- **A *Domain* is characterized by a Conceptual Schema in OOHDM**
  - The only hot spot allowed are classes that are flagged as allowing specialization during the framework instantiation

Navigation Reuse in OOHDM-Frame



Lessons Learned

- Industry still uses few methods
- Tool support
- Graphical notations can work
  - careful choices!
- Specialized vocabularies help
- Models must be used properly

Conclusions

- Progress has been made
- Challenging tasks remain
- Opportunity from Semantic Web
- Interaction with other areas

Thanks! Questions?

- Journal of Web Engineering
  - http://www.rintonpress.com/journals/jwe
- http://www.webengineering.org
- OOHDM Wiki
  - http://www. oohdm.inf.puc-rio.br:8668
- My email
  - dschwabe@inf.puc-rio.br