

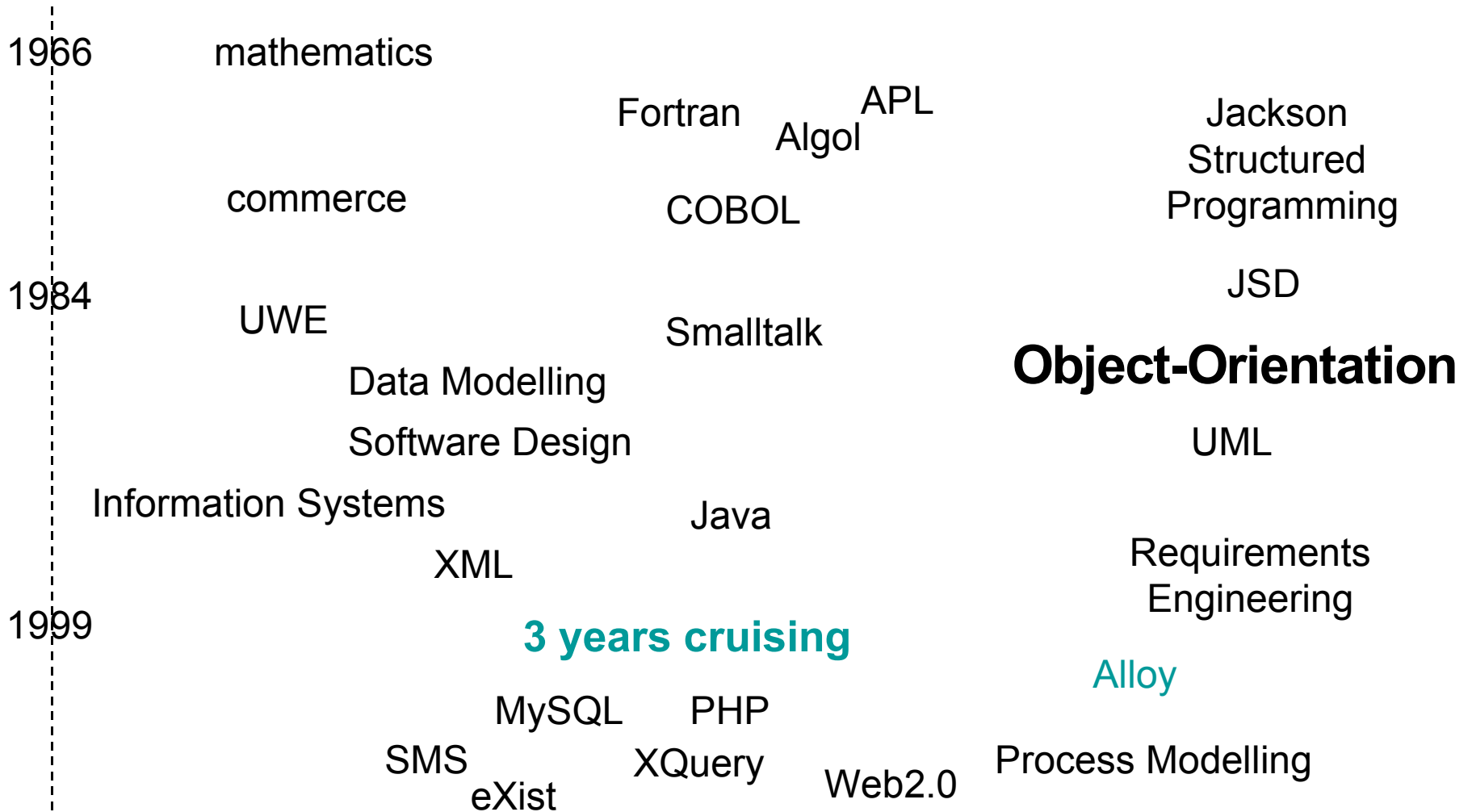
# eXist in the Faculty

Chris Wallace  
Senior Lecturer  
School of Information Systems  
University of the West of England, Bristol, UK

XML Prague  
18<sup>th</sup> June 2006

# My tag cloud

New Zealand



# UWE

- UWE
  - University of the West of England, **Bristol**
  - Ex Polytechnic
  - 23,000 students
  - 1,000 teaching staff
  - 9 Faculties (Law, Business, Art Media and Design,.. CEMS)
  - Central student records, payroll, finance
  - Administrative systems devolved to Faculties

# Computing, Engineering and Mathematical Sciences

(and Information Systems)

- 2000 students
- 450 taught modules
- 100 Programmes (Courses)
- 350 staff
- 150 organisational units (schools, functional groups, committees, research groups)

# Two kinds of data

- XML Database (FOLD)
  - Structured data with controlled update
  - Data is well-defined and hence easily searchable
  - Aim is to provide a single point of definition of faculty information by integrating a range of data sources
- Content Management System (DOMUS)
  - Implemented using Plone
  - Any staff member can set up a section, store documents, create discussion groups..
  - Aim is to support collaboration e.g.
    - Documentation on this project
    - Information Systems Field information
    - Faculty Board Agenda and Minutes

# FOLD – Faculty OnLine Data

- FOLD Scope
  - Descriptions of all resources within the Faculty
  - Teaching and assessment management
  - Intranet for staff
  - Public site students, prospective students, the public
- In the past
  - Information distributed over word documents, spreadsheets, access databases, SQL database, flat text files, LDAP....
- Aims
  - To support distributed data ownership
  - To provide a web of data within and between systems
  - To support organisational processes
  - To support student choices
  - To improve information quality and availability

# FOLD current statistics

- **Code**
  - XQuery 4000
  - XSLT 4000
  - XSD 300 (one schema)
  - CSS 200
  - PHP 100 (SMS)
- **Installation**
  - Solaris on Sun server quad-486 2GB
- **Sites**
  - One internal, 1 public
  - about 40 pages
- **Information System development**
  - CW (6 person/months)
  - 3 students (12 person/months)
  - Clerical support
  - Tech support
  - Phase allocation:
    - Project (20%)
    - Code (20%)
    - Data – gathering, conversion, cleaning (60%)



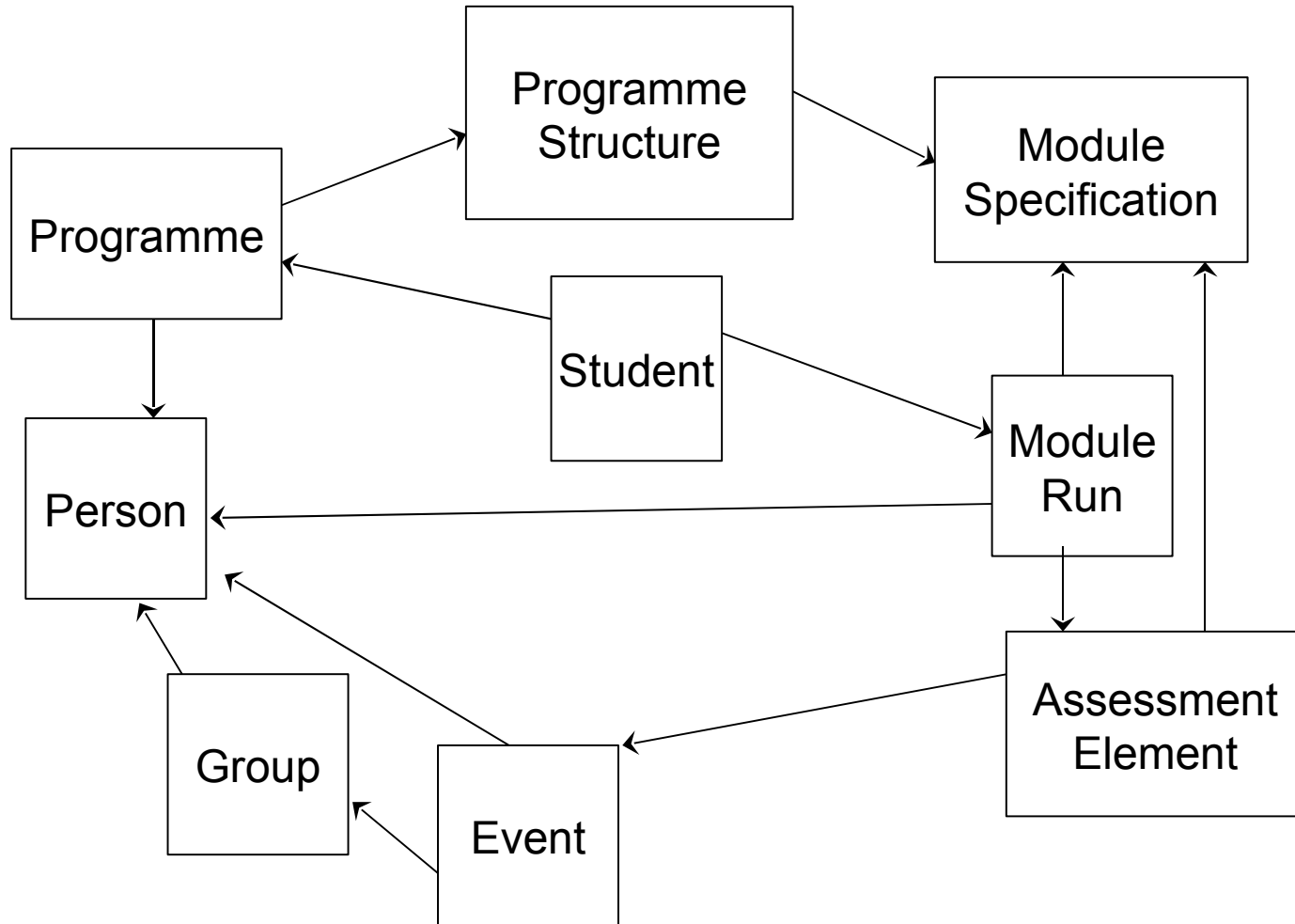
# Composites

- Documents as Coherent Composites
  - Hierarchy of composition relationships
  - Structured relationships – and/or expressions
  - Ordered relationships
  - Subtypes
  - Structured text – restricted HTML
  - Composite has organisational meaning – ownership, editorship, viewing

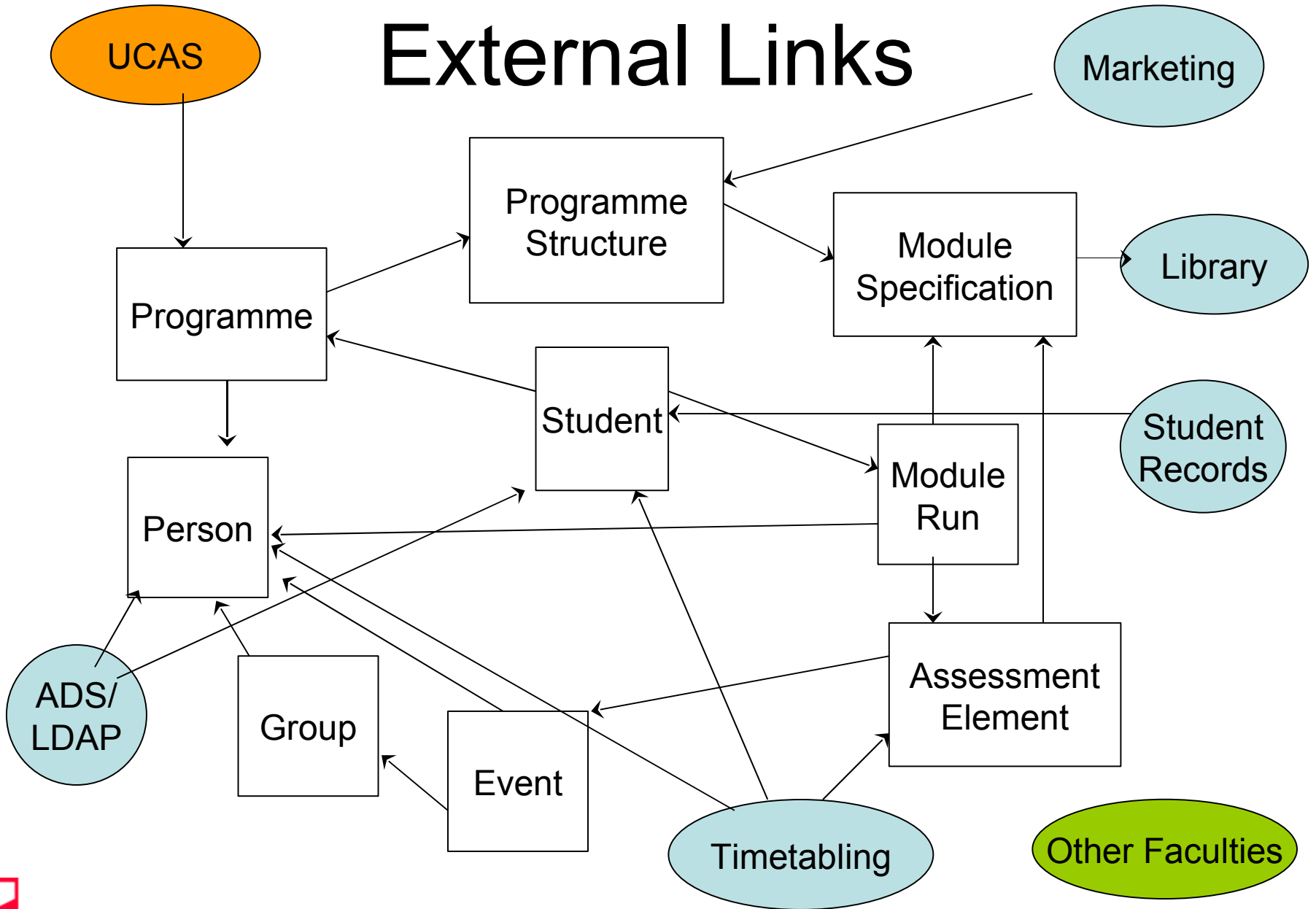
# A Network Database

- Associations
  - Links implemented with shared identifiers
  - Uni-directional
  - Either direction if no salient direction
  - Identifiers are ‘natural’
    - to reduce Real World/model gap
    - to support unanticipated linking
- Issues
  - Duplicate keys
    - Issue in the real world too, so social mechanisms to resolve
    - ‘realm’ (cf namespace in the schema domain)
  - Key Change
    - No deletion – past is always true
    - Multiple primary keys

# Internal Links



# External Links



# FOLD Entity Types (extract)

Entity Type	Identifier	No of instances	Document Map	No of documents (change/year)	Document Type
Module Specification	ModuleCode/Version	450	one each	450 (80)	complex structure
Module Run	ModuleCode/Year/Runno	460	one per field/year	6	table
Module assessments	ModuleCode/Year/Runno/Element No	800	one per field/year	6	Table
Examination	ModuleCode/Year/Exam	420	one per year	1	
Student numbers	Date/ModuleCode	450 * 4	one per date	5	table
Award types	PrimaryAward	8	one only	1	simple structure
Programmes	ProgrammeCode/Year	100	one per year	1	table
Programme Structure	ProgrammeCode/Pathway/Version	110	one each	110 (20)	complex structure
Organisational structure	GroupName	100	several per major group	60	simple structure
Events	EventGroup/EventID	300	all events in a group	50	simple structure
Staff	Name	400	per responsibility	5	table with reps
Training	Name/Course	200	one only	1	table
Training Courses	Course	40	one only	1	table
ucasKey words	UCASCode/Keyword	4000	one only	1	table
UWE calendar	Date	365	one only	1	table
SuggestedHours	Level	5	one only	1	simple structure
Entity Type metadata	DatasetName	20	one only	1	table
System Configuration	Faculty	1	one only	1	table

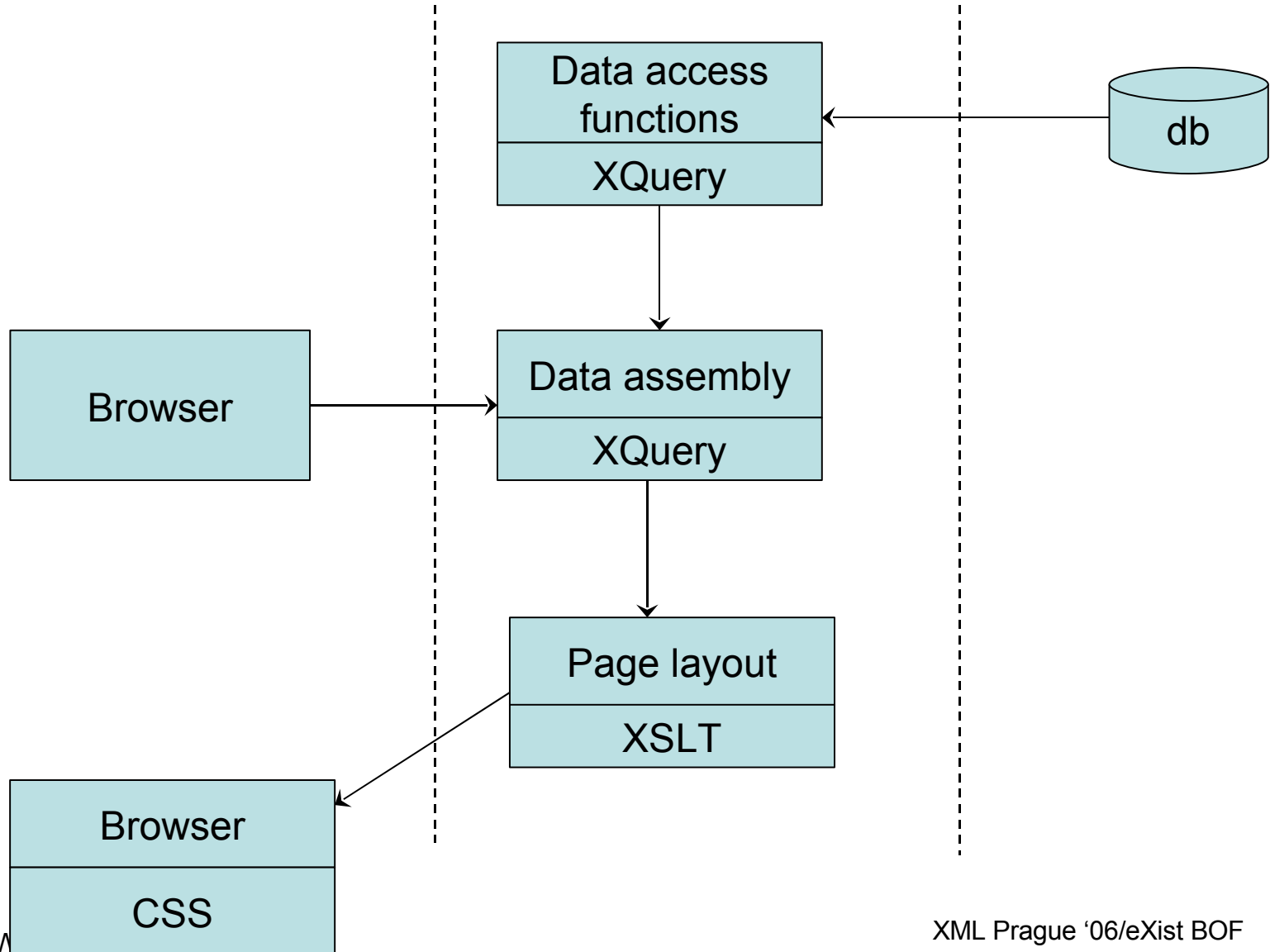
# Development Approach

- ‘Agile Information Systems development’
  - Concurrent development of data content and software
  - Concurrent learning about the application domain and how to represent it
    - Data / Structure / Code dialectic
  - Incremental take-on of content
  - Both data and code need re-factoring
    - Just the job for XSLT and xmldb:store
  - Staying in the shallows...

# Staying in the shallows

- **Jetty installation**
- **Staying a couple of snapshots behind the leading edge**
- **No attributes (in base data)**
  - Never can decide which to use
  - Hard to edit in Word or Excel
  - Meta-data is complex too
- **No schemas**
  - Too much work to keep in line during development
  - 'XML is self-describing'
  - Users create documents by example –prototyping principal
  - Model fidelity more important than model integrity
- **No user indexes**
  - Save this optimisation for the future
- **No forms (yet)**
  - All data edited as XML
  - Word and Excel 2003 as editors
  - XForms (with FormFaces ?)

# Application architecture



# Sample Design/Coding issues

- Computed Views
- Value Inheritance
- Document Versioning
- Expression evaluation

# Computed Views

```
xquery version "1.0";
```

```
(: Create Coursework Cover sheet :)
```

```
declare namespace request="http://exist-db.org/xquery/request";
```

```
declare namespace transform="http://exist-db.org/xquery/transform";
```

```
import module namespace modprog =
```

```
"http://www.cems.uwe.ac.uk/fold/modprog" at "modprog.xqm";
```

```
let $moduleCode := request:request-parameter("moduleCode", ""),
```

```
    $year := modprog:currentYear(),
```

```
    $runNo := request:request-parameter("runNo", "1"),
```

```
    $elementNo := request:request-parameter("elementNo", "1"),
```

```
    $element := modprog:courseworkElement($moduleCode, $year, $runNo,  
$elementNo),
```

```
    $coversheet := doc("/db/fold1/prod/courseworkCover.xsl")
```

```
return
```

```
    transform:transform($element, $coversheet, ())
```

```

declare function modprog:courseworkElement($moduleCode, $year, $runNo,
    $elementNo) {
let $mod := modprog:moduleSpecification($moduleCode,$year),
    $run := modprog:moduleRun($moduleCode,$year,$runNo),
    $elementRun := modprog:elementRun($moduleCode,$year,$runNo,"B", $elementNo),
    $elementSpec := $mod/Assessment/FirstAttempt/Components/ComponentB/Element
        [position() = $elementNo],
    $dueDate := $elementRun/DueDate,
    $lateDate := events:workingDays($dueDate,10),
    $returnDate := events:workingDays($dueDate,20),
    $componentWeight := $mod/Assessment/Weighting/ComponentWeightB,
    $weightInComponent := xs:decimal($elementSpec/Weight),
    $weightInModule := round($weightInComponent * $componentWeight div 100),
    $load := modprog:load($mod/Level),
    $hrs := round(xs:decimal($mod/UWERating) div
        xs:decimal($load/Credits) * $weightInModule div 100
        * data($load/Hours)),
    $year := modprog:academicYear(modprog:currentYear())

```

return

```
<CourseworkElement>
  <ModuleCode>{$moduleCode}</ModuleCode>
  {$mod/Title}
  <Year>{$year}</Year>
  <RunNo>{$runNo}</RunNo>
  {$run/ModuleLeader}
  {$run/ActingModuleLeader}
  {$run/Tutors}
  {$run/InternalModerator}
  {$run/ExternalExaminer}
  <Component>CW</Component>
  <ElementNo>{$elementNo}</ElementNo>
  {$elementSpec/Description}
  <SuggestedHours>{$hrs}</SuggestedHours>

  <WeightInComponent>{$weightInComponent}</WeightInComponent>
  <WeightInModule>{$weightInModule}</WeightInModule>
  <DueDate>{string($dueDate)}</DueDate>
  <LateDate>{string($lateDate)}</LateDate>
  <ReturnDate>{string($returnDate)}</ReturnDate>
</CourseworkElement>
```

# Value Inheritance

- Value inheritance is inheritance in a network of instances
  - c.f. Class inheritance in a network of classes
- Where path is hard-coded
  - SQL provides coalesce()
  - XQuery node sequence ignores nulls

```
declare function modprog:currentYear() {  
  let $sessionYear := request:get-session-attribute("currentYear")  
  let $defaultYear := modprog:config()/currentYear  
  return  
    ($sessionYear,$defaultYear,"2005")[1]  
};
```

# Versioning

- Changes every year
  - Explicit versioning
  - Every year has a document
    - Operational data – runs of modules
- Changes infrequently
  - Applicable version at any time must be inferred
    - Specifications of modules and programmes

```
declare function modprog:moduleSpecification($moduleCode,$year) {
```

```
(: get the set of versions prior to and including this year :)
```

```
  let $modspecs :=
```

```
  collection('/db/fold1/modules/specs/')/ModuleSpecification
```

```
    [ModuleCode=$moduleCode] [Version <= $year]
```

```
(: select the version which is the latest version :)
```

```
  let $modspec := $modspecs[Version = max($modspecs/Version)]
```

```
  return $modspec
```

```
};
```

# Expression evaluation

- Module Pre-requisites
  - A and B and (C or D)
- Display
- Evaluation
- Reduction

```
<ModuleCombination>  
  <andexp>  
    <Code>A</Code>  
    <Code>B</Code>  
    <orexpr>  
      <Code>C</Code>  
      <Code>D</Code>  
    </orexpr>  
  </andexp>  
</ModuleCombination>
```

# Expression Evaluation

```
declare function andor:exp($e, $list) {  
  if (name($e) = "Code")  
  then andor:code($e,$list )  
  else if (name($e) = "orexp")  
    then andor:orexp($e, $list)  
    else if (name($e) = "andexp")  
      then andor:andexp($e, $list)  
      else true()};
```

```
declare function andor:andexp($e,$list) {  
  every $exp in $e/* satisfies andor:exp($exp,$list)};
```

```
declare function andor:orexp($e, $list) {  
  if (count($e/*) > 0 )  
  then some $exp in $e/* satisfies andor:exp($exp,$list)  
  else true()};
```

```
declare function andor:code($e, $list) {  
  $e = $list};
```

# Breaking Down walls

- Shared identifiers
  - ISBN
  - in-house identifier standards
- Standard schemas
  - icalendar data from events for Outlook
  - KML for GoogleEarth
  - Voice+XML for Opera to provide text-to-speech
- Multimedia/ multimodal
  - Graph generation using Graphviz
  - Tag clouds
  - SMS (texting)

# Lessons Learnt

- Establish naming conventions
  - XML elements, functions, arguments, scripts, documents, collections, XQuery/XSLT variables...
- Use Micro-schemas
  - Small reusable fragments
    - <role>
    - <event>
- XSLT for schema transformation

# Little things to sort out

- Executing a server process
  - e.g. GraphViz for graph generation
- Relative pathnames in:
  - doc() and collection()
  - module inclusion
  - path to stylesheet in a processing instruction
- cycles in module inclusion
- deep-equals (for unit tests in XQuery)
- input(url) – returning plain text
- custom error page

# Bigger things to do

- eXist
  - LDAP authentication
  - WebDav with Microsoft Office
  - Improvements to the Java Client as an IDE
- Development Approach
  - Understand the eXist architecture
  - A decent XML database case tool
    - Global model mapping to multiple schemas
  - Code management
- Application
  - Authorisation
  - XForms
  - Process modelling
    - Predicates over event trace, not statemachine
  - Plone CMS - eXist balance
  - eXist performance

# Takeaways

- XQuery and the eXist Native XML database are the most fun I've had since Smalltalk
  - actually more because it delivers useful systems
- 'Take the code to the data, not the data to the code'
  - PHP/ Java
    - loading and unloading XML <> structures
  - XQuery/XSLT
    - Direct operations on the XML data
    - Move to XForms
- 'It's the data, stupid'
  - A successful Information system depends on:
    - Data quality – model fidelity more than model integrity
    - Handling multiple, sometimes inconsistent, views of the world