XML, Objects, Relational Databases, Oh My!

- The current bleeding edge of information systems is a confluence of relational database management systems, object-oriented programming, and XML
- Learning curves for each of these technologies can be steep (depending on what you already know) — but for now we'll talk about them breadth-first: figure out how everything fits together and connects first, then proceed with learning more about each component individually…ultimately, we want to come up with an actual, living, breathing piece of software

The Situation

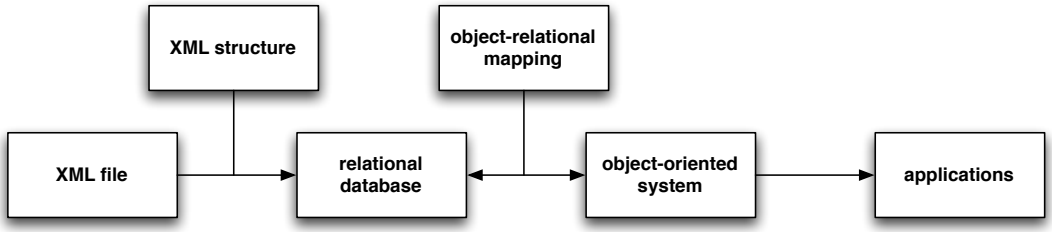
- Relational databases are the prevalent standard for storing, managing, and querying large amounts of data
- Object-oriented programming is the prevalent paradigm for developing applications (client or server, Web or non-Web)
- XML is gaining momentum as a "common ground" for sharing information
- How can we build object-oriented software that accesses relational databases, containing information that is initially available in XML format?

Impedance Mismatch at Multiple Levels!

- Relational model ≠ object model ≠ XML — but they'll all hold the same information!
- One approach was to unify the models — e.g., develop an object-oriented database or define a "serialized" object format — but the reality is that each paradigm has its strengths and a momentum of its own
- Another option is to *manually map* information across these boundaries — but this is tedious and error-prone, and leads to software that is not reusable
- The current thinking: automation, at all levels

Two key technologies come to our aid here:

- The XML standard includes mechanisms that can *specify* the structure of a given XML file (DTD, XSD)
- Techniques/conventions have been developed for translating a relational structure to an object structure — an activity called *object-relational mapping* or ORM

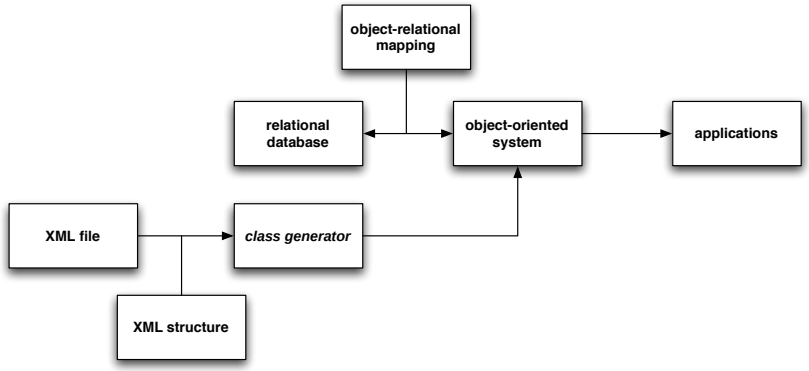


Specifying XML Structure

- Originally, the XML standard provided for a *document type definition* (DTD) file, which specified the structure and content for different types of XML files
- This standard has since evolved into XML Schema, using the *XML schema definition* (XSD) format which is itself in XML
- XML files can specify the DTD or XSD to which they conform; validators have been developed to compare a given file against a DTD or XSD to see whether that file "complies" with the claimed structure

From XML to Objects

- One can make an observation that an XSD is analogous to defining a *class* in an object-oriented programming language
- Thus, an individual XML file can be viewed as containing one or more *instances* of the class defined by an XSD
- This observation has manifested itself in the existence of programs that reads an XSD file and *generates* Java classes that correspond to that file — Castor, JAXB, XGen, XMLBeans, and many more



- So, our flow changes slightly: we still want to load XML files into a relational database, but instead of direct loading, we represent the XML data as objects and classes first, and *then* store the objects in the database
- Note how the object-oriented system becomes the unifying representation; it thus serves as the *conceptual* or *canonical data model* for the overall system

From Objects to Relations

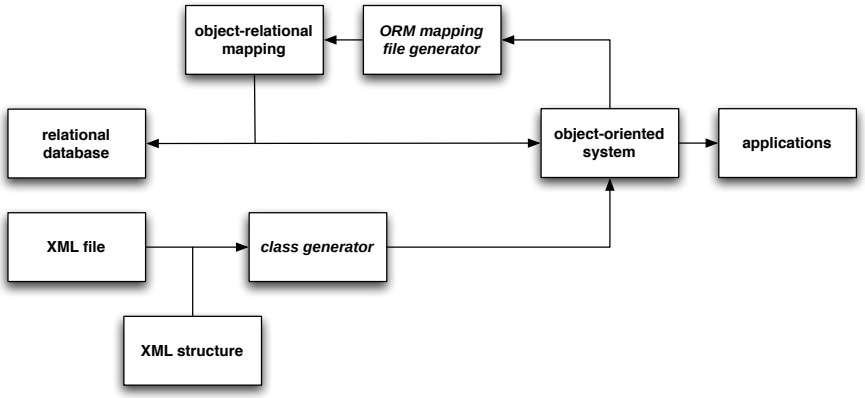
- An initial approach toward automated object-relational mapping followed this process:

  1. Define the objects/classes and relations
  2. Build a *mapping file* between the objects and relations

- Tools for doing this include Hibernate, JDO, iBatis
- This is easier than writing custom code to read/write objects from a relational database, but it still can be a pain — if either the class or relation changes, the mapping file has to be modified as well

Making ORM Easier

- Since a conceptual binding exists between objects and relations anyway, why not *embed* the mapping in one of the two representations?
- It turns out that, for Java, a widely-used tool called *Xdoclet* can fulfill just that purpose
- Xdoclet can read *annotations* in Java code, then generate new files (Java, XML, HTML, whatever) based on those annotations
- With Xdoclet, we can automate the generation of a mapping file: just change the annotations in the source

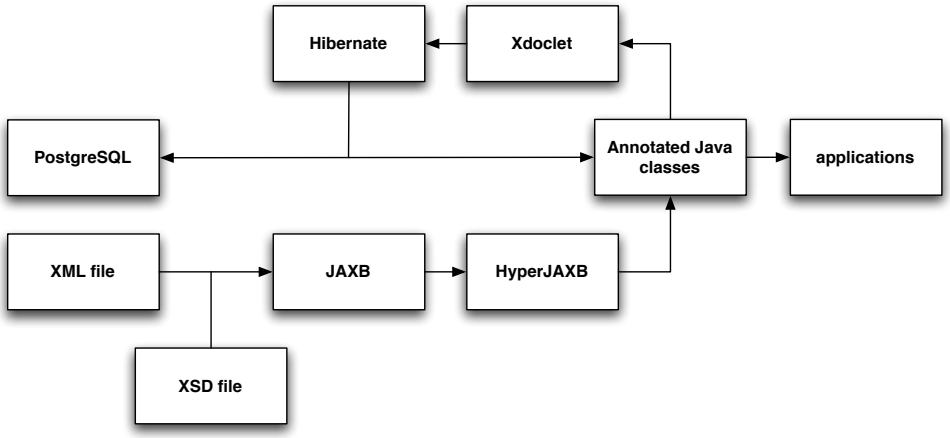


- So now what's left? Well…recall that the classes that we are mapping are themselves automatically generated from the XML schema definition — that means, if the XML schema changes, we would have to regenerate the classes and reinstitute the ORM annotations
- Unless…unless someone has developed a tool that does *both* — generate the classes *with* the ORM map

The Final Setup

- It turns out that a tool for almost fully automating the entire process, from XML to objects to relations, does exist — HyperJAXB bridges the gap between JAXB, Hibernate, and Xdoclet!
- As you may have inferred from previous mentions of JAXB, Hibernate, and Xdoclet, the unifying object-oriented environment for all of this is Java
- For the bottom layer — the relational database — we can pretty much use any RDBMS that has a *JDBC driver* — a standard Java interface to a relational database

- So now we have a target architecture:



- "Applications" are anything that can be written in Java — servers, Swing applications, Web applications, etc.

In a Nutshell…

General Technology	Description	Specific Example(s)
XML	Extensible format for sharing structured data	Everyone and their mother
XML Schema	Mechanism for specifying the information structure of XML-formatted data	Formerly DTD, now officially **XSD**
XML-object mapping	Conversion from XML documents to classes and objects	**JAXB**, Castor, XGen, XMLBeans
object-oriented programming	Programming paradigm that captures state and behavior	**Java**, C++
object-relational mapping	Conversion to/from the relational model from/to objects	**Hibernate**, JDO, iBatis
relational database	Persistent repository for data	**PostgreSQL**, Oracle, MySQL

- Key supporting tools are:
  - *Xdoclet* — automatically generates Hibernate ORM files based on annotations in Java code
  - *HyperJAXB* — automatically adds said annotations to Java code generated by JAXB
- And one more thing — you don't really want to invoke this sequence of tools manually every single time, so we also need a build system for this whole thing; for Java development, *ant* is widely used for this purpose