

Data Access Strategies in Modern Business Application Development

Glenn Engstrand

<http://www.transitionchoices.com/cgi-bin/article.pl?id=18>

January 21, 2008

Abstract

A large majority of the software in any business application is concerned with creating, reading, updating, and deleting the complex data that models the operation of any business. This paper surveys how modern business application software development attempts to solve this problem by formulating the data access strategy that is most appropriate and efficacious to the problem being solved by the software.

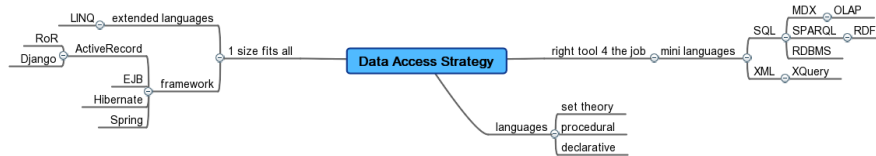


Figure 1: Mind Map

Choose Your Weapon

The source code of software is written in a particular programming language or languages. The first step in formulating a proper data access strategy is to choose the most appropriate language(s). There are different types of programming languages. Code written in a procedural language tends to look like a complex series of instructions such as a recipe; move this over there, write this to a file, add up these numbers. Declarative languages look more like a collection of facts; if this is true then so is that. A very interesting type of language, when it comes to accessing complex data, is all about describing sets of things. Think of it as a sort of textual description of a Venn Diagram.

In order to understand how to select the most appropriate programming language or languages, you must also understand how modern software is organized. Most modern systems are organized in an MVC or Model, View, Controller pattern where the user sees and interacts with the data through a series of views. A controller is used to coordinate the views and to bind each view to its appropriate model. The model is the code that accesses the data.

Another important thing to understand and consider is how complex data is organized. Data is typically organized either in a hierarchical fashion, as a large collection of two dimensional tables, or as a single table that has many dimensions. A database is the schema (or meta-data or structure) of the data and the data itself. Examples of hierarchical databases are LDAP (short for Lightweight Directory Access Protocol), XML (short for eXtensible Markup Language) databases or object oriented databases [14]. There is a particular flavor of XML that is most suitable for capturing data. Called RDF or Resource Description Framework, this variant of XML organizes everything in terms of triplets; the subject, the predicate, and the object [9]. A relational database organizes the data as lots of two dimensional tables. An OLAP (short for On Line Analytical Processing) database has only one table (traditionally called a cube) but that cube will have many dimensions. Each dimension will also be organized in a hierarchy. Hierarchical databases are best when the data itself is hierarchical in nature (e.g. file folders). Relational databases are most appropriate in operational systems (e.g. purchasing books). OLAP databases are used by analysts to optimize some aspect of a business (e.g. spreadsheets that pivot to track your best and worst performers over time) [7].

The choice of language(s) must also reflect the preferences and skill-sets of the developers who will be working on the application. It is imprudent to choose a language that none of your developers are familiar with. If you take anything away from this article, it should be this. When it comes to language selection, there are two camps of developers. There are those developers who believe that *one size fits all* and there are those developers who want to pick the *right tool for the job*.

One Language to Rule Them All

Those who believe in the one size fits all strategy tend to code in only one language. In order to avoid huge amounts of boilerplate code doing the same kinds of create, read, update, and delete functionality, You must either extend your programming language to make it easy to both access and to operate on data or you use a framework that does the data access for you.

Language Extensions

.NET is a runtime environment and application stack for writing so called managed applications which means that the memory management is handled by the runtime instead of the programmer. Any programming language that can be compiled into the Intermediate Language that runs in the .NET virtual machine is considered to be a .NET language. There are many .NET languages including VB.NET and C#. The LINQ Project is a codename for a set of extensions to the .NET Framework that encompass Language INtegrated Query, set, and transform operations. It extends C# and Visual Basic with native language syntax for queries and provides class libraries to take advantage of these capabilities [5]. LINQ can be used to query either relational or XML databases. The .NET languages are optimized for expressing business rules. LINQ is optimized for expressing sets of data. You have the best of both worlds in a single language.

Framework Extensions

Instead of extending the language, you can extend the class library to provide an ORM or Object Relational Mapping. One of the first attempts at providing an ORM is EJB or Enterprise Java Beans. There are two types of EJB, session beans and entity beans. There are two flavors of entity beans, CMP (Container Managed Persistence) and EMP (Entity Managed Persistence). For CMP, the EJB developer doesn't have to know or write any SQL as it is the application container that does all of that for you [3].

One of the first lightweight ORMs was Hibernate. EJB coding has a lot of dependencies. Any application written to use EJB was tightly coupled to that approach. Also, there is a lot of boilerplate that had to be used when producing or consuming EJBs. Hibernate coding didn't have a lot of dependencies. The code is loosely coupled so you could switch to a different persistence medium if that was preferred. There is no boilerplate code when working with Hibernate because Hibernate worked with POJOs (Plain Old Java Objects) whereas EJB did not [4].

Another very cool ORM is Spring. Spring is an impressive confluence of different J2EE technologies. Spring has a great IoC/DI framework for rapid web application development. Spring has a great ORM that also works with POJOs. If you don't like Spring's ORM, then you can use Spring to plug into Hibernate. Spring also has a great AOP (short for Aspect Oriented Programming) component too. Spring is available for both J2EE and .NET programming [11].

One of the newest and most rapidly growing strategies is to use the ActiveRecord design pattern. This pattern was developed by Martin Fowler. Basically, any object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data is based

on the ActiveRecord design pattern [1]. The two most popular and viable web application developer frameworks delivering on ActiveRecord are Ruby on Rails and Django.

Ruby on Rails is a popular web application server for Ruby developers. You create the relational database schema and run various ruby utilities for a wizardly generation of various scaffolding and boilerplate code [10].

Django is a very popular web application development framework for Python developers. You author classes in Python that represent the data model and then you run the django utilities to generate the scaffolding code for everything including the SQL code to create the database schema [2].

It Takes a Village

If you come from the right tool for the job camp, then you will have no problem using multiple languages. Typically, this approach involves using a general purpose programming language to specify the business rules and a language that is good for expressing sets of data for the data access parts. This second language is usually called a *mini-language* and is either embedded within the code of the main language or called from that code (e.g. stored procedures in a relational database).

Introducing SQL

The mini-language of choice for relational data access is SQL. Short for Structured Query Language, SQL has been the only way to access relational databases for decades and relational databases are very nearly the ubiquitous way for storing operational business data of any significant size and complexity. There are SQL commands to create, drop, grant, revoke, delete, update, and insert various objects in the database but it is the select command that is the most interesting for accessing data. With the select command, you can write queries that perform intersection and union operations on the data from the two dimensional tables, both vertically and horizontally. The select command specifies the field list and can include the following clauses; from, where, group by, having, and order by. Multiple select statements can also be unioned together to specify a single result set. Queries can also be nested in other queries [13]. All the ORM approaches discussed previously use SQL under the covers. It's just that the application developer doesn't experience coding the SQL.

Extending SQL

SQL is traditionally used on relational databases but there is an extension to SQL for OLAP databases called MDX. Short for Multi-Dimensional eXpressions, MDX extends the familiar SQL syntax with these

extra set specifications, typically in curly braces. In an MDX query, a member in the hierarchy for every dimension (or axis) of the cube must be specified. The columns and rows are specified in the field list portion of the select statement. Query axes specify the edges of a result set. Slicer axes specify how to filter the data. In a normal SQL statement, the field list specifies the columns of the result set and each row represents a record or records from some number of tables in the database. In an MDX query, the schema of both columns and rows are actually two dimensional tables in and of themselves and the table representing the results are actually aggregates of what is being measured in the specified slice of the cube. The from clause specifies what cube to use. The where clause specifies what is being measured [6].

Another extension to SQL for RDF databases is called SPARQL. This extension adds some new clauses to the SQL syntax and puts curly braces in the where clause. The prefix clause associates namespaces with their full IRI. The field list determines what predicates will become columns in the result set. If the construct clause is present, then the result set is shaped like another RDF graph. If the ask clause is present, then the result set is actually a boolean indicating whether or not there are any results that match. If the describe clause is present, then the result set contains only the resource information of the results that match. Result sets are filtered using the filter command in the where clause [8].

Querying XML Databases

The mini-language of choice for accessing XML data is XQuery. This standard is an extension of another standard called XPath. There are five parts to an XQuery; let, for, where, order by, and return. The let clause initializes a variable to the XML document being queried. The for clause specifies what is being iterated over and binds those iterations to variables. The where clause specifies the filtering criteria. The return clause specifies what the result set should look like [12].

Selection Factors

So, which way should you go? If your development team is fairly inexperienced and the requirements of your application don't ask for much more than create, read, update, and delete style access, then consider the one size fits all approach. If your's is a .NET shop, then go the LINQ route; however, both Hibernate and Spring are available options for you. If your coders know Java, then consider either Hibernate or Spring as your ORM. The latest version of EJB looks a lot like Hibernate. I would not even consider earlier versions of EJB. If your coders know Ruby, then go the Rails route. If your coders know Python, then go the Django route. If your coders know SQL or if you have lots of complex reporting requirements, then you will have to embrace SQL to some

degree. MDX really is the best choice for accessing OLAP databases. If you need to query an RDF database, then use SPARQL. This is going to work only if there exists a SPARQL interpreter for your main programming language. At the time of this writing, SPARQL is fairly new. If you need to query an XML database, then use XQuery. LINQ is also an option for querying XML databases from the .NET environment.

References

- [1] MARTIN FOWLER, *Active Record*, <http://martinfowler.com/eaCatalog/activeRecord.html>
- [2] LAWRENCE JOURNAL-WORLD, *The Web Framework for Perfectionists with Deadlines*, <http://www.djangoproject.com>
- [3] SUN MICROSYSTEMS, *Enterprise JavaBeans Technology*, <http://java.sun.com/products/ejb>
- [4] RED HAT MIDDLEWARE, LLC, *Relational Persistence for Java and .NET*, <http://www.hibernate.org>
- [5] MICROSOFT CORPORATION, *The LINQ Project*, <http://msdn2.microsoft.com/en-us/netframework/aa904594.aspx>
- [6] MICROSOFT CORPORATION, *Multidimensional Expressions Reference*, <http://msdn2.microsoft.com/en-us/library/ms145506.aspx>
- [7] BUSINESS APPLICATION RESEARCH CENTER, *The OLAP Report: What is OLAP?*, <http://www.olapreport.com/fasmi.htm>
- [8] ERIC PRUD'HOMMEAUX, ANDY SEABORNE, *SPARQL Query Language for RDF*, <http://www.w3.org/TR/rdf-sparql-query>
- [9] IVAN HERMAN, RALPH SWICK, DAN BRICKLEY, *Resource Description Framework*, <http://www.w3.org/RDF>
- [10] DAVID HEINEMEIER, *Ruby on Rails*, <http://www.rubyonrails.org>
- [11] SPRINGSOURCE, *Spring Framework*, <http://www.springframework.org>
- [12] LIAM QUIN, *W3C XML Query*, <http://www.w3.org/XML/Query>
- [13] OMAGESPHERE, *SQL Database Reference Material*, <http://www.sql.org>
- [14] LIAM QUIN, *Extensible Markup Language*, <http://www.w3.org/XML>