XMLMetadata

Test Strategy for Chapter 18: XML Metadata

Testing the orm XML metadata of Chapter 18 requires the following components:
*an object model, manifested as java classes
*a mapping of the classes to a relational schema
*data to populate the model
*a test to verify that the data is correctly stored and retrieved via jdo We plan to use a single object model, a single mapping, a single data set, and a single test to perform as much of the testing as possible. Where necessary, these will be supplemented with a special purpose model, mapping, data, or test. The primary test object model is called the Company model; the primary test case is called the Completeness test.

Chapter18TestComponents lists the XML tags and whether each can be tested by the Completeness test and Company model.

The UML diagram for the Company model is at CompanyModelUml.

All tests must be run both for application and datastore identity.

We plan to create a schema generator utility that will generate database-specific DDL from the mapping metadata (see DataBaseGen).

XML Metadata ToDo

Activity wiki page	Description	Who	Expected Completion Date
Chapter18TestComponents	Fill in further details of alternate testing strategies on Chapter18TestComponents	Michelle Caisse	?/05
Ch18TestSpecs	Specify Completeness test and additional tests	TBD	?/05
MetadataMappings	Develop mappings	TBD	?/05
–	Create data to populate the model	TBD	?/05
–	Implement Completeness test and other tests required	TBD	?/05
DataBaseGen	Write XML metadata to DDL schema generator	TBD	?/05
PersistentInterfaces	Update XML metadata to use factory; add persistent interfaces and properties to standard metadata; add CompanyFactory implementations for persistent interfaces	Craig Russell	9/05

Notes on mapping metadata

*SimpleMapping
*SimpleMappingQuestion
*JoinCondition
*EmbeddingClasses
*EmbeddingQuestion
*RelationshipsOneToMany
*RelationshipsManyToOne

Notes from jdo-tck-ext weekly meetings on xml metadata

November 19:

Agenda:

1. Discuss present Company model to see if it is suitable for use for mapping tests.

2. Review all metadata elements/attributes and map against Company model. 3. Take a look at Chapter 15 examples.

It looks like the Company model is suitable for many of the mapping tests, but some of the extended mapping elements will need either their own mapping /schema or an entirely new model. We reviewed the model along with the metadata to see how well the model met the requirements.

To support embedded sub-classes, we need to define a different schema:
add subclass for US Address postal code for US embedded derived class is embedding a common enough case to put in separate schema? Probably not. We concluded that we should have a separate model/mapping/schema to handle embedded subclasses. The Company model is sufficient for the basic embedding/serialization.

To support multiple relationships between the same model classes, we will add fundingDepartment, fundedEmployees to the Employee/Department model.

To support subclassing of one-one relationships, and multiple one-one relationships between different model classes, we will add MedicalInsurance, Dental Insurance as subclasses of Insurance and have one-one relationship between Employee and medicalInsurance and between Employee and dentalInsurance.

More types should be added, but we agreed to add String[], Collection<String> to another model. These are only a small part of a list of extended types that need model/mapping/schema support.

Company is mostly used by query and extent tests.

May want to use Company for generating schema. This involves adding enough information to generate schema from the mapping (table, column, index names, jdbc types) but we want to avoid putting sql-types into the mapping, as these are database dependent.

Must support schema as modified by vendor. The TCK needs to specifically state that the schema distributed with the TCK can be changed at will. We also need to specify which parts of the mapping can be changed.

We need to define navigate test as a base test that can populate a database with a specific set of test data and verify that the objects were properly stored.

create graph of company schema using input file to specify the objects and fields commit
get new pm, start tx get company by id navigate to entire graph. verify contents of the datastore against the input file.

We need separate schema, object model, mapping for persistent interface types. We won't add persistent interfaces to the Company model.

Testing null fields. Add middle name as nullable String to Person; have two Person constructors one with middle name one without

We should look at Chapter 15 examples to see if we can rewrite them to use the Company model and mapping. Most test organizations would at least verify that the examples compiled and parsed. The easiest way to do this would be to use examples from the Company model.

We got about halfway through the list of elements and attributes. We will continue at the next conference call.

Since next week is Thanksgiving, we will not have a TCK conference call. But we should meet the following Friday. I'll send out an announcement of the meeting.

Craig

December 3:

Attendees:

Michelle Caisse, Victor Kirkebo, Michael Bouschen, Patrick Linskey, Craig Russell.

Reviewing the Chapter18Test.html document, starting with foreign-key element.

Not clear whether we need to or how we are supposed to test foreign-key delete-action or update-action. Need a spec update to clarify exactly what these attributes are. Patrick was volunteered to provide a reference to the Kodo documentation that describes what they do with these attributes.

Underspecified dependent objects' life cycle behavior. Should transition to deleted if reference is nullified or removed from a collection. This topic interacts with foreign-key delete-action.

Embedded element needs clarification 18.13.1. Kodo uses collection embedded element to describe separate table with elements in it, whereas Craig thought that the spec clearly called out that embedded elements are supposed to be mapped to the same table as the rest of the fields.

We will resume the discussion next week at the map element.

Craig

December 10:

We completed reviewing the attached document as a base for deciding how much work is involved in writing the TCK tests for the metadata chapter.

The next steps are to write a specification for the base mapping metadata test, and to update the document to provide more detail on the kind of test /mapping/metadata needed for the tests that cannot be done with the "Completeness Test".

We discussed Matthew's proposal to add a Map to the Company model, and agreed that a simple map of primitives made sense for the model, but more complex types (pc key, primitive value; primitive key, pc value; pc key, pc value) should have its own model and test.

We will meet again this coming Friday, same call number, same time 9:00 AM PST.

866 230-6968 294-0479#
865 544-7856

This will be the last meeting this year, as the following Friday is Christmas Eve.

Craig

December 16

Hi,

Michael is experimenting with a [BeanFactory] package available as part of the springframework [sic]. It allows you to define a graph of [JavaBeans] via an external xml file.

We're considering using this package as part of the TCK, as we discussed, to implement the Chapter 18 Completeness Test. The idea is to construct a graph of objects, persist them, and then make sure they are stored and retrieved correctly. Using a tool like BeanFactory will simplify the test.

In order to exploit this package, we need to make some adjustments to the Company model that we have discussed as part of the tck. These changes allow us to implement a graph compare function that verifies that the object graph created and committed is able to be retrieved correctly.

We need to define an interface DeepEquality in org.apache.jdo.tck.util that is implemented by all the classes in the company model that defines a method called deepEquals that does a deep comparison of "Collection of another DeepEquality". The signature of the methods are boolean deepEquals (DeepEquality other). We also want a convenience method that "deepEquals" two collections of DeepEquality.

The model for this pattern is interface Comparable that defines a method int compareTo(Object other).

The idea is that to compare deep collections, the deepEquals method does:

boolean deepEquals(DeepEquality other) {
Company c = (Company) other; this.name.equals(other.name); ...
DeepEqualityHelper.deepEquals(this.depts, other.depts); ...
}

class DeepEqualityHelper {
boolean deepEquals(Collection mine, Collection other) {
List myList = new ArrayList(mine); Collections.sort(myList); List otherList = new ArrayList(other); Collections.sort(otherList); for (i = 0; i < myList.size()) {
if (!(DeepEquality)myList.get.deepEquals(otherlist.get) return false;
}
return true;
}

boolean shallowEquals(Collection mine, Collection other) {
List myList = new ArrayList(mine); Collections.sort(myList); List otherList = new ArrayList(other); Collections.sort(otherList); return myList.equals(otherList);
}
All classes in Company must define equals to delegate to compare and must implement Comparable and DeepEquality.

Craig Russell

December 17:

Javadogs,

Agenda:

1. Review Spring Framework BeanFactory approach to testing mapping.
2. Issue: how to stop deepEquals in case of circular references, e.g. an Employee is her own manager.

b. Issue: BeanFactory creates instances of LinkedHashSet which are not supported by FOStore.

c. Issue: How to create different xml files (data load and metadata) by creating diffs. We expect that around 100 different files will be needed, of which a substantial portion will be identical to the base mapping files. We want to just look at diffs not the entire mapping file (to avoid duplication of bugs in the mappings).

d. Additions to EqualityHelper (renamed DeepEqualityHelper):

Compare two doubles, floats, "close enough". Compare BigDecimal for equality ignoring scale.

class EqualityHelper {
boolean deepEquals(DeepEquality me, DeepEquality other); boolean deepEquals(Collection, Collection); boolean deepEquals(Map, Map); boolean shallowEquals(Collection, Collection); boolean closeEnough(Object, Object); // other variants tbd boolean closeEnough(double, double); // others needed? boolean equals(Object, Object);
}

Note that closeEnough needs to compare two numbers and check that they are within some percent of each other. An absolute epsilon is not sufficient. For example, closeEnough(0.001, 0.002) should return false but closeEnough(1000000.001, 1000000.002) should return true.

interface DeepEquality {
boolean deepEquality(Object other);
}

2. Think about this: should we use the same strategy for queries? That is, define the persistent object graph via BeanFactory xml and run queries against it? Queries are probably the biggest part of the functionality of mapping that needs testing.

Craig