WebObjects and Scala - Divergent Paradigms

Although WebObjects and Scala work remarkably well together, there are some rough edges.
Here we'll take a look at conflicting paradigms of WebObjects and Scala.

Foundation

WO Collections vs. Scala Collections

One of the primary advantage of Scala is its very rich and modern set of collection classes - List, Map, etc.
These enable the use of functional programming techniques such as pattern matching, comprehensions and functional mapping & filtering. They are also thread-safe for concurrent computations.

Although Scala allows for conversions between its collection classes and Java collections, it's preferable to use the Scala foundation classes universally in a Scala WebObjects application to avoid unnecessary extra noise.

Scala 2.9 introduces parallel collections. These may offer a more powerful way of dealing with "astronomical" database relationships.

Key Value Coding vs. Strong, Static Typing

Scala is a strongly typed static language. As a result WebObjects *Key Value Coding* (or **KVC** for short) is out of place in a Scala world.
At one time WO's *KVC* was one of my favourite features of the dynamically typed Objective-C language. However even in the strongly typed Java language *KVC* started to appear out of place if not awkward. It is passe.

Scala has syntactic sugar that makes "( )" of methods optional.
So accessing "studio.movie.name" via *KVC* would be identical (if not simpler) in Scala without.

Many of the perceived benefits of weakly typed, dynamic languages may not be valid anymore.

Java:

```
String studioName = (String) valueForKeyPath("movie.studio.name");
```

Scala:

```
val studioName = movie.studio.name
```

KVC Interfaces

The following are a list of impacted API

- `NSKeyValueCoding`
- `NSKeyValueCodingAdditions`
- `NSKeyValueCoding.ErrorHandling`
- `EOKeyValueCoding`
- `EOKeyValueCodingAdditions`
- `EOKeyValueCoding.ErrorHandling`
- `NSValidation`

Other API

As with *KVC*, other weakly typed WebObjects APIs don't fit comfortably in a strongly typed world like in Scala.

Examples

- `pageWithName()`
- `valueForBinding()/takeValueForBinding()`
- `createInstanceWithEditingContext()`

EOF

EOF is notoriously single-threaded.
As a result EOs and business logic are unlikely to receive any significant benefit using Scala.

Locking and Synchronizing Shared Mutable Data vs. Actor Model

The use of shared data and locks doesn't prohibit concurrency in anyway. It is just considerably harder to get right.
The Actor model of concurrency which is used by Scala offers a simpler, radical alternative. Say goodbye to locks...

EOSharedEditingContext

Technically, though the shared editing context is supposed to store only "read-only" data, one is not disallowed from editing those EOs.
This is why the shared editing context should be turned-off in a multi-threaded, concurrent WebObjects application.

Shared Object Cache

EOF by default will share row-level snapshots of objects. This very much goes against the Actor model paradigm of concurrency which vehemently avoids the use of mutable shared data.
It may be one reason why EOF is not considered (or used) in a multi-threaded way.

EOF's shared object cache model may not have been a bottleneck in the days of monolithic or dual-core CPUs but now with multi-core servers commonplace, it may well be.

Other

Imperative Programming vs. Functional Programming

The WebObjects API is presented in an *imperitive*, *object-oriented* style.
Though Scala can be used imperatively like Java, it is vastly more powerful when used as a functional programming language.
Functional programming concepts (like *closures*) and style (e.g: *chaining*) have become popular in several modern languages and libraries such as *Ruby*, *JavaScript*, *jQuery*, *Prototype*, *Python* and *Perl* amongst others.

Scala Wildcard '_' vs. Java Wildcard '*'

Scala's wildcard character is '_' unlike Java's '*' wildcard.
In addition to making reading Scala awkward at first, it also means that it's not such a great idea to have Scala classes prefixed with '_'. E.g: EO generated classes like `_Talent`.