



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences



Migrating a Struts/Java eMail Application to Lift/Scala

by

Christoph Knabe

<http://prof.beuth-hochschule.de/knabe>

Thomas Fiedler

Talk at the ScalaDays 2010, 15./16.04.10, Lausanne



My Background (Christoph Knabe)

- 1971 Learning Assembler, and Algol 60 on Zuse 22
- 1979 Diploma degree at Bonn University
- 1981-1990 Industry automation at PSI GmbH, Berlin
- 1990-now Professor at TFH Berlin, renamed to BHT Berlin
- Admin of faculty web server fb6.beuth-hochschule.de
- Doing Java Web Development
- Mostly teaching software project in media informatics studies
- Scala is my 14th strong programming language.
- GUIs are not my strong point.



Content

Our Application

Central Exception Reporting

Reporting Exceptions in Untraditional Requests (AJAX, Comet)

Reporting Exceptions in Background Actors

Transaction Management

Singleton Objects

Mapper Entity Definition

Mapper Highly Configurable

Mapper Queries

Mapper CRUD pages

Mapper vs. JPA/Hibernate

HTML Templating

[Testing, Development Environment, Logging, ScalaSnippet]: Only in Wiki

Conclusion



Our Application *Teacher News*

- E-mails students about teacher-related events.
- Students can subscribe their actual teachers.
- Faculty administration can send messages about every teacher, *e.g. on illness*.
- Teacher can send messages about himself, *e.g. on moving an exercise deadline*.
- Currently about 1000 subscribers and 120 teachers.
- Since 2003
- Running with Struts 1, Tomcat 5.5, JPA 1/Hibernate, PostgreSQL 8, Java 6, Debian Linux 5.



Central Exception Reporting with Lift on Traditional Requests

- **Goal:** No necessity to catch an exception.
 - If propagating, it should be reported in a user-friendly format.
- **Default behavior:**
 - If propagating through to Lift, the stack trace is displayed:

Exception occurred while processing/test

```
Message: java.lang.ArrayIndexOutOfBoundsException: Array index out of range: -1
    com.lehrkraftnews.snippet.NewsSnippet.showTestForm(NewsSnippet.scala:280)
    sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    ...
```

- **Lift hook in method** `Boot.boot` e.g.

```
LiftRules.exceptionHandler.prepend {
  case (runMode, req, exc) =>
    PlainTextResponse(exc.toString)
}
```



Our Solution for Central Exception Reporting (Traditional Requests)

- Return nice global exception page by this in `Boot.boot`:

```
LiftRules.exceptionHandler.prepend {  
  case (runMode, req, exc) => ExceptionResponse(runMode, req, exc)  
}
```

- The necessary class `ExceptionResponse` can be defined as on the next slide.
- Demo:**
 - Make the H2 database file `lehrkraftnews.data.db` read-only.
 - Log-in as a teacher at http://localhost:8080/user_mgt/login
 - Go to `Edit User`, modify something, and confirm `Edit`.
 - The failure will be reported on a special exception page (shown after the next slide).



```
/** Generates a full HTML exception page*/
case class ExceptionResponse(runMode: RunModes.Value, request: Req, exc: Throwable)
extends HeaderStuff with NodeResponse {
  val docType = ResponseInfo.docType(request)
  val code = 500 //Internal Server Error
  override val renderInIEMode = S.ieMode
  val out: Node = {
    val excContent: Node = (<lift:surround with="default" at="content">
      <div><h1>Error Occured</h1>
        <p>We could not execute your request <pre>{request.uri}</pre> due to the following error:</p>
        <p class="errorBox">{multex.Msg.getMessages(exc)}</p>
        <p/>
        <h3>The error occured at location:</h3>
        <pre>{multex.Msg.getStackTrace(exc)}</pre>
        <table>
          <tr><th>RunMode:</th> <td>{runMode}</td></tr>
          <tr><th>Request:</th> <td>{request}</td></tr>
        </table>
      </div>
    </lift:surround>)
    S.containerRequest match {
      case Full(httpRequest) => //Traditional request
        S.render(excContent, httpRequest)(0) // (0) takes first Node of NodeSeq
      case _ =>
        excContent
    }
  }
}
```

With Notice about S.render



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

[Test](#) [Start](#) [Nachrichten](#) [Abonnements](#) [Hilfe](#)

knabe@beuth-hochschule.de [Logout](#) [Edit User](#) [Change Password](#)

Error Occured

We could not execute your request

/user_mgt/edit

due to the following error:

```
org.h2.jdbc.JdbcSQLException: Die Datenbank ist schreibgeschützt The database is read only, SQL statement: UPDATE person SET lastname = ? WHERE id = ? [90097-112]
```

The error occured at location:

```
org.h2.jdbc.JdbcSQLException: Die Datenbank ist schreibgeschützt  
The database is read only; SQL statement:  
UPDATE person SET lastname = ? WHERE id = ? [90097-112]  
    at org.h2.message.Message.getSQLException(Message.java:107)  
    at org.h2.message.Message.getSQLException(Message.java:118)  
    at org.h2.message.Message.getSQLException(Message.java:77)  
    at org.h2.message.Message.getSQLException(Message.java:153)  
    at org.h2.engine.Database.checkWritingAllowed(Database.java:1760)
```




Reporting Exceptions in Untraditional Requests

Exceptions can occur when Lift processes untraditional (AJAX or Comet or Actor) requests.

■ **Default behavior:**

- Untraditional request exceptions are logged by Lift
- They are not catchable by `LiftRules.exceptionHandler`.
- The client doesn't report anything. If you don't click any link, it displays after a while a confirmation dialog with the text **The server cannot be contacted at this time.** During the wait period the client retries the request about 3 times.

■ **Lift hook:**

- `S.addAround(lw: LoanWrapper)` offers pre- and post-processing the Lift request processing. It captures all kinds (Full HTML, AJAX, Comet, and Actor) of requests. Reporting Notice



Our Reporting of Untraditional Request Exceptions

- On untraditional request:
 - Catch and store exception in `LiftSession`
- On next traditional request:
 - Put an error message by `S.error` into the Lift framework,
 - which will be reported by the `default` template.
 - Afterwards we delete the stored exception.
- The code is:
 - In method `Boot.boot`
call `S.addAround(new ExceptionReporting)` .
- **Demo:**
 - Make the H2 database file `lehrkraftnews.data.db` read-only. Then try the Comet form <http://localhost:8080/news/add> and after filling in the message press "Speichern". On any next click you will see the delayed error message from the Comet exception.



Error:

org.h2.jdbc.JdbcSQLException: Die Datenbank ist schreibgeschützt The database is read only; SQL statement: INSERT INTO nachricht (inhalt,gueltig_bis,betriff_person_id) VALUES (?,?,?) [90097-112]

Warning:

- Error during former asynchronous action. See above.

Alle Nachrichten

Gültig bis	Lehrkraft	Inhalt
		1
30.04.2010	Knabe---, Christoph	test 25
30.04.2010	Knabe---, Christoph	Lift is super.
30.04.2010	Knabe---, Christoph	at scaladays
30.04.2010	Knabe---, Christoph	See you again at ScakaDays 2011.
27.04.2010	Knabe---, Christoph	still moving on
21.04.2010	Knabe---, Christoph	test 26
14.04.2010	Knabe---, Christoph	ScalaDays will begin tomorrow.
14.04.2010	Knabe---, Christoph	ScalaDays will begin tomorrow.



```
/**Assures that exceptions are reported at the next traditional request.*/
class ExceptionReporting extends LoanWrapper {

  /**Will be invoked by Lift when processing a request.*/
  override def apply[T](doLiftProcessing: => T): T = {
    val result = if(ExceptionReporting.isTraditionalRequest(S.request)){ //Traditional request
      //Now report the last asynchronous exception, if exists:
      var oldAsyncExceptionMsg = ExceptionReporting.takeOut // from LiftSession
      oldAsyncExceptionMsg match {
        case Empty =>
        case Full(messages) => _reportError(messages)
        case x => _reportError(x.toString)
      }
      doLiftProcessing
    }else{ //Untraditional (AJAX, Comet, or Actor) request:
      try{
        doLiftProcessing // Let Lift do normal request processing.
      } catch {
        case e: Exception => {
          log.warn("Exception when processing an AJAX/Comet/Actor request:", e)
          ExceptionReporting.store(e) // into LiftSession
          throw e //Will not be reported by Lift, but if we did not throw, we had to return result of the erased type T!
        }
      }
    }//if isTraditionalRequest
    log.trace("After request. Result = " + result)
    result
  }
  ...
} //ExceptionReporting
```



Reporting Exceptions in Background Actors

- **Concerns:** Actors not tied to a user interface element.
 - Typically: `act` method contains a `loop` with a `react` case block, in which occurs the processing of various messages accepted by the actor.
- **Default behavior:**
 - Exception in the `react` block interrupts the `loop`, and `Actor`. Stack trace is logged to `System.err`.
- **Our case and solution:**
 - Actor is a modification of `net.liftweb.util.Mailer.MsgSender`. Mail sending triggered by UI action \Rightarrow Send a failure message to a `CometActor` tied to the UI form.
 - The `CometActor` shows failed eMail addresses one by one.
 - **Demo:** Make the H2 database file `lehrkraftnews.data.db` writable. Then try the Comet form <http://localhost:8080/news/add> and after filling in the message press "Speichern". You will see succeeding in white, and failing eMail addresses in red, one by one.



Nachrichten: Hinzufügen

Gültig bis:

content:

Nachricht als E-Mail versenden?

E-Mail-Versand Abgeschlossen

Versand erfolgreich:

knabe@bht-berlin.de

knabe@tfh-berlin.de

knabe@beuth-hochschule.de

Versand Fehlgeschlagen:

barack.obama@beuth-hochschule.de

angela.merkel@bundesdeutschland.de

willi.brandt@beuth-hochschule.de



Transaction Management

- **Goal:** As in every database application,
 - We want to encapsulate each business logic operation into a transaction.
- **Default behavior:** Lift running in auto-commit mode.

- **Lift offers ready solution** (in `Boot.boot`):

```
//For making each request a database transaction:  
S.addAround(DB.buildLoanWrapper(List(DefaultConnectionIdentifier)))
```

- **What does the following mean?**
 - INFO: No Transaction manager found - if your webapp requires one, please configure one.
 - Concerns a JTA Transaction Manager, which is only necessary for distributed transactions.



Singleton Objects

- **Why?** Lift offers many singleton objects \neq dependency injection

Object	Purpose
<code>LiftRules</code>	Configuring Lift
<code>SHtml</code>	Generating function-rich XHTML elements.
<code>S</code>	Access to the actual request and response, as well as to <code>LiftSession</code> attributes.
<code>Mailer</code>	Services for sending mails.

- **Disadvantages** by using these object names:
 - Cannot substitute implementations; not important for `LiftRules`
- **Advantages**
 - In Mapper: Using same name for entity and its DAO is very ergonomic.



Mapper Entity Definition

- **Scope:** Traditional OR mapping + serving CRUD pages.
- **Entity requirements:** No POJO!
 - Entity extends base Mapper class, mixes in traits.
 - Attributes extend `MappedDatatype` classes.
- **Example.** Message with attributes `concerningUserId`, `content`, and `expirationDate`:

```
class Message extends LongKeyedMapper[Message] with IdPK {  
  def getSingleton = Message // Get the message DAO object  
  object concerningUserId extends MappedLongForeignKey(this, User)  
  object content extends MappedTextarea(this, 2000)  
  object expirationDate extends MappedDateTime(this)  
}
```

For entity `m`: Message read attribute `content` by `m.content.is`, set its value by `m.content(value)`.



Mapper Highly Configurable

- **Attributes** can overwrite many methods, which provide defaults.
 - E.g. you can change the attribute's database column name, its representation as HTML, its column name in displayed tables, and its validation.

```
class Message extends LongKeyedMapper[Message] with IdPK { ...
  object concerningUserId extends MappedLongForeignKey(this, User){
    override def dbColumnName = "concerns_person_id"
    /**Showing the attribute on CRUD pages.
     * Instead of the ID the name of the concerning person will be shown.*/
    override def asHtml =
      Text(User.find(this).map(_.fullCommaName).openOr("n.a."))
    override def displayName = "Teacher"
    override def validSelectValues: Box[List[(Long, String)]] =
      Full(User.findAll.map(u => (u.id.is, u.lastName.is)))
  } ...
}
```



Mapper Queries

- **Typesafe** queries like in a dedicated, compiled OQL:
 - Many helper classes to formulate queries
 - Queries look understandable, 100% checked by compiler. E.g.:

```
/**Returns all users in the role Teacher in ascending order concerning last name.**/  
def allTeachers = User.findAll(  
  By(User.userRole, Teacher),  
  OrderBy(User.lastName, Ascending)  
)
```



Mapper CRUD Pages

- **No Scaffolding.** Simply mix in the `CRUDify` trait into a DAO object:

```
object Message extends Message
  with LongKeyedMetaMapper[Message]
  with CRUDify[Long, Message]
{ ... }
```

- The CRUD functions are accessible under the following URIs:

Function	URI
List all	/message/list
Show one	/message/view/ <i>id</i>
Edit one	/message/edit/ <i>id</i>
Create one	/message/create
Delete one	/message/delete/ <i>id</i>

- I like it, as generated code would be difficult to debug.



Mapper vs. JPA/Hibernate

- **Advantages of Mapper**
 - Configuration in Scala:
Avoids language-rupture; Gives compile-time checking.
 - Highly configurable by overwriting attribute object methods
 - Gives CRUD editing
 - Elegant, type-safe criteria API by pure-Scala means.
- **Advantages of JPA/Hibernate**
 - Can manage POJO entities.
 - Using annotations for meta info is well-readable.
 - Standardized.



HTML Templating

Lift processes HTML templates surrounding them by and including other templates or generated content from Scala snippets. Very powerful! E.g.

User editing template:

```
<lift:AdminUserSnippet.edit
form="POST">
  Nachname: <br/>
  <user:lastname/> <br/>
  Rolle: <br/>
  <user:role /> <br/>
  <user:save /> ...
</lift:AdminUserSnippet.edit>
```

In method `AdminUserSnippet.edit` you bind values or active UI components to the tags in the HTML:

```
bind("user", xhtml,
  "lastname" -> text(user.lastName.is,
    currentUserVar.is.lastName(_)),
  "role" -> select( roleNames.toSeq,
    Full(user.userRole.is.toString),
    a => currentUserVar.is.userRole(a.toInt)
  ),
  "save" -> submit("Save", doSave))
```

- Works very well.
- Better to avoid string literals by generating symbolic constants from HTML!



Code Statistics

- For the functions:
 - Subscribe at teachers
 - Send new own message
 - Show messages of all teachers
 - Show messages of selected teacher.
- Code sizes in Lines:

■ With Lift:	With Struts
■ 362 LoC HTML/XML	948 LoC JSP/XML
■ 1,629 LoC Scala	4,447 LoC Java
■ 1,991 LoC Sum	5,395 LoC Sum
■ Lift 37% LoC	Struts 100% LoC



Conclusion about Lift

■ **Negative**

- Lack of documentation:
You have to understand the Lift source code.
Even that is not fully documented, e.g. in `SHTML`.

■ **Positive**

- Very powerful
- Compact, compiler-checked notation
- Many working defaults
- Highly configurable: All wishes we had, were realizable.
- Behavior of Lift/Scala is stable.

■ **State**

- *Teacher News* ported, planned to go into production.