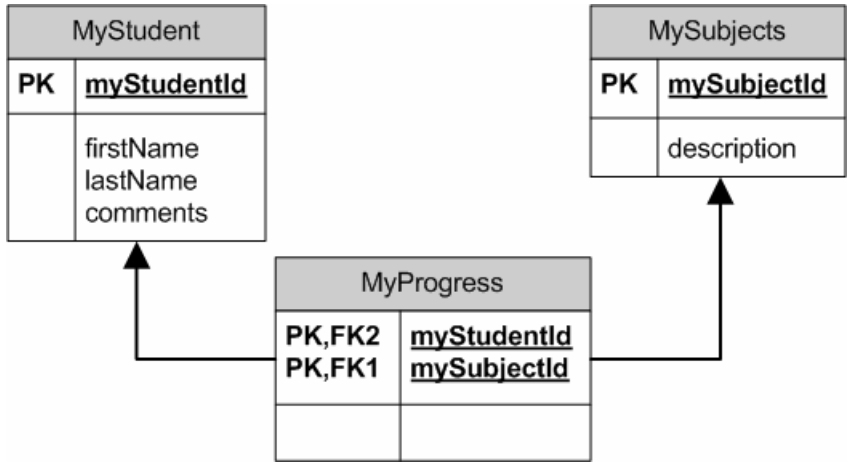
A First OFBiz Application

ERP-systems, Aalborg University, Information Systems

Klemens Schwarz (Upper Austria University of Applied Sciences, Hagenberg)

Building a complete Application with OFBiz
(in order to understand the structure of the OFBiz layers)

- What are we going to build?
  - A simple site for administrating the students' progress (*„StudentProgress"*).
- Data Model:
  - Two main tables
  - One connecting table



- Services:
  - Create new students
  - Create new connections between students and their finished subjects/courses
  - (we assume, that subjects already exist within the application)

During your work it may be necessary to reuse existing data tables and services, but maybe also create new ones.

Create the data model

1. Define the tables in entitygroup.xml in the /entitydef-directory

```
<?xml version="1.0" encoding="UTF-8"?>
<entitygroup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/entitygroup.xsd">
    <entity-group group="org.ofbiz" entity="MyStudent" />
    <entity-group group="org.ofbiz" entity="MySubject" />
    <entity-group group="org.ofbiz" entity="MyProgress" />
</entitygroup>
```

2. Define the colums and primary and foreign keys in entitymodel.xml

```
[...]
<entity entity-name="MyStudent"
            package-name="org.ofbiz.studentProgress"
            title="Entity for storing data about students">
    <field name="myStudentId" type="id-ne"></field>
    <field name="firstName" type="id"></field>
    <field name="lastName" type="id"></field>
    <field name="comments" type="comment"></field>
    <prim-key field="myStudentId"/>
</entity>
[...]
```

Required change to some files

- component-load.xml in /hot-deploy

```
[...]
<load-component component-location="${ofbiz.home}/hot-deploy/studentProgress"/>
[...]
```

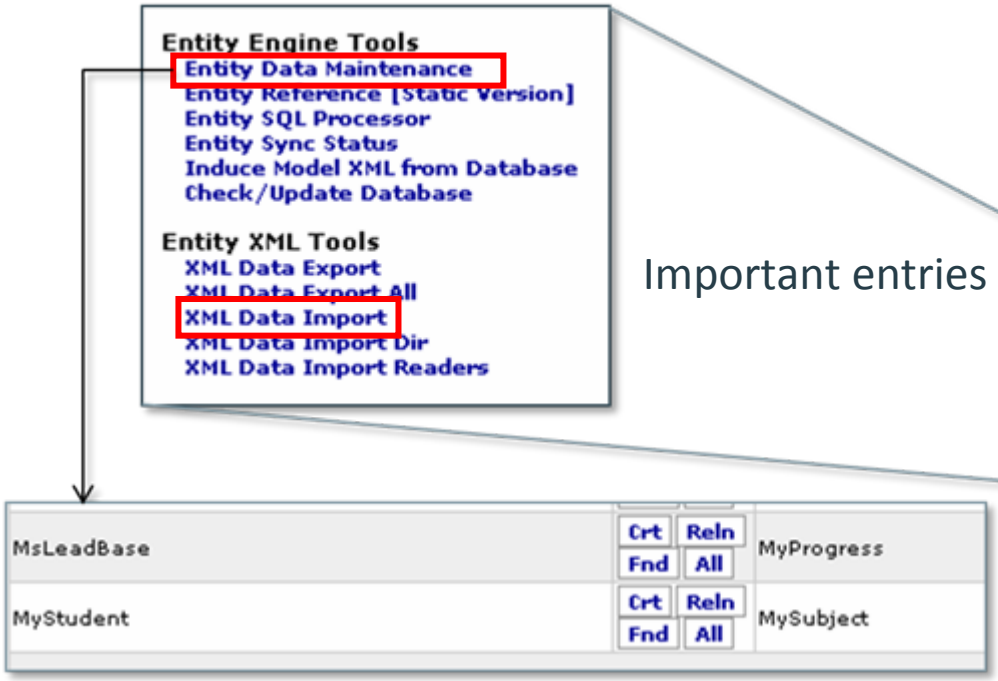
  - ofbiz-component.xml in /hot-deploy/studentProgress

```
<ofbiz-component [...]>
    [...]
    <entity-resource type="model" reader-name="main" loader="main"
          location="entitydef/entitymodel.xml"/>
    <entity-resource type="group" reader-name="main" loader="main"
          location="entitydef/entitygroup.xml"/>
    [...]
</ofbiz-component>
```

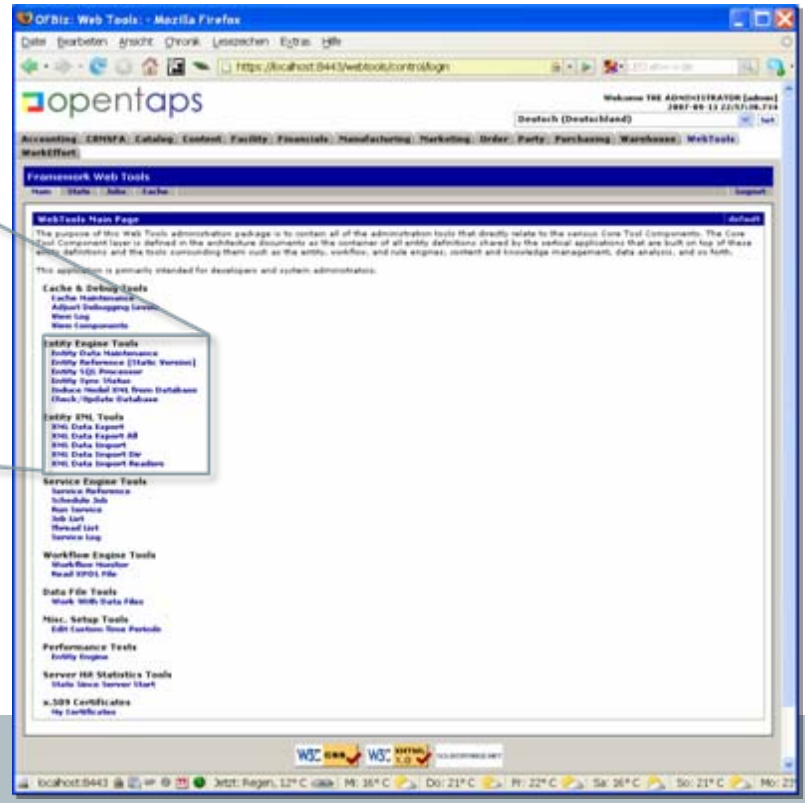
- *You can copy these files from existing applications.*
- *Starting ofbiz.jar would already generate these tables (but nothing else yet).*

Insertion: The OFBiz Webtools application

- On http://localhost:8080/webtools you will find an important application for your development process.



Important entries



kschwarz@fh-hagenberg.at

Creating the business logic

- 2 Steps:
  1. Define your services in general in an XML file (with parameters and where to find the service, Java, OFBiz minilang or other scripting languages).
  2. Implement the service either in Java, OFBiz minilang, or others.

Step 2 might be the more difficult part.

Step 1: Define Services

- Inside /servicedef directory: services.xml

```
<services […]">
    <description>Student Progress Services</description>
    <!-- this will be implemented in Java -->
    <service name="createMyStudent" engine="java"
        location="org.ofbiz.studentProgress.MyServices" invoke="createMyStudent">
        <description>Create a new student</description>
        <auto-attributes mode="IN" entity-name="MyStudent" include="nonpk"
           optional="true"/>
        <attribute name="myStudentId" mode="OUT" type="String" optional="false"/>
    </service>
    <!-- this will be implemented in OFBiz minilang -->
    <service name="createMyStudentSubject" engine="simple"
        location="org/ofbiz/studentProgress/MyServices.xml"
           invoke="createMyStudentSubject">
        <description>Create a MyProgress entity which links a student and a
           subject</description>
        <auto-attributes mode="IN" entity-name="MyProgress" include="pk" optional="false"/>
    </service>
</services>
```

Step 2a: Java service in /src directory

```
public class MyServices {
  public static Map createMyStudent(DispatchContext dctx, Map context) {
     GenericDelegator delegator = dctx.getDelegator();
     try {
        String studentId = delegator.getNextSeqId("MyStudent");
        Debug.logInfo("myStudentId = " + studentId, module);
        GenericValue student =
          delegator.makeValue("MyStudent", UtilMisc.toMap("myStudentId", studentId));
        student.setNonPKFields(context);
        delegator.create(student);
        Map result = ServiceUtil.returnSuccess();
        result.put("myStudentId", studentId);
        return result;
     } catch (GenericEntityException ex) {
     return ServiceUtil.returnError(ex.getMessage());
  }
}
```

The services written in Java code have to be compiled (using ant and a build.xml file → see next slide).

Build Java files with ant (or similar)

Example output:

```
Buildfile: D:\opentaps\opentaps\hot-deploy\studentProgress\build.xml
init:
clean-lib:
   [delete] Deleting directory D:\opentaps\opentaps\hot-
deploy\studentProgress\build\lib
prepare:
    [mkdir] Created dir: D:\opentaps\opentaps\hot-deploy\studentProgress\build\lib
classpath:
classes:
    [javac] Compiling 1 source file to D:\opentaps\opentaps\hot-
deploy\studentProgress\build\classes
jar:
      [jar] Building jar: D:\opentaps\opentaps\hot-
deploy\studentProgress\build\lib\ofbiz-party.jar
BUILD SUCCESSFUL
Total time: 42 seconds
```

→ **Results: class- and jar-files used by the framework**

Step 2b: Minilang service in /script directory

```
<?xml version="1.0" encoding="UTF-8" ?>
<simple-methods xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/simple-methods.xsd">
    <simple-method method-name="createMyStudentSubject" short-description="Create a
        Student-Subject relationship“ login-required="false">
        <make-value entity-name="MyProgress" value-name="newEntity"/>
        <set-nonpk-fields map-name="parameters" value-name="newEntity"/>
        <set-pk-fields map-name="parameters" value-name="newEntity"/>
        <create-value value-name="newEntity"/>
    </simple-method>
</simple-methods>
```

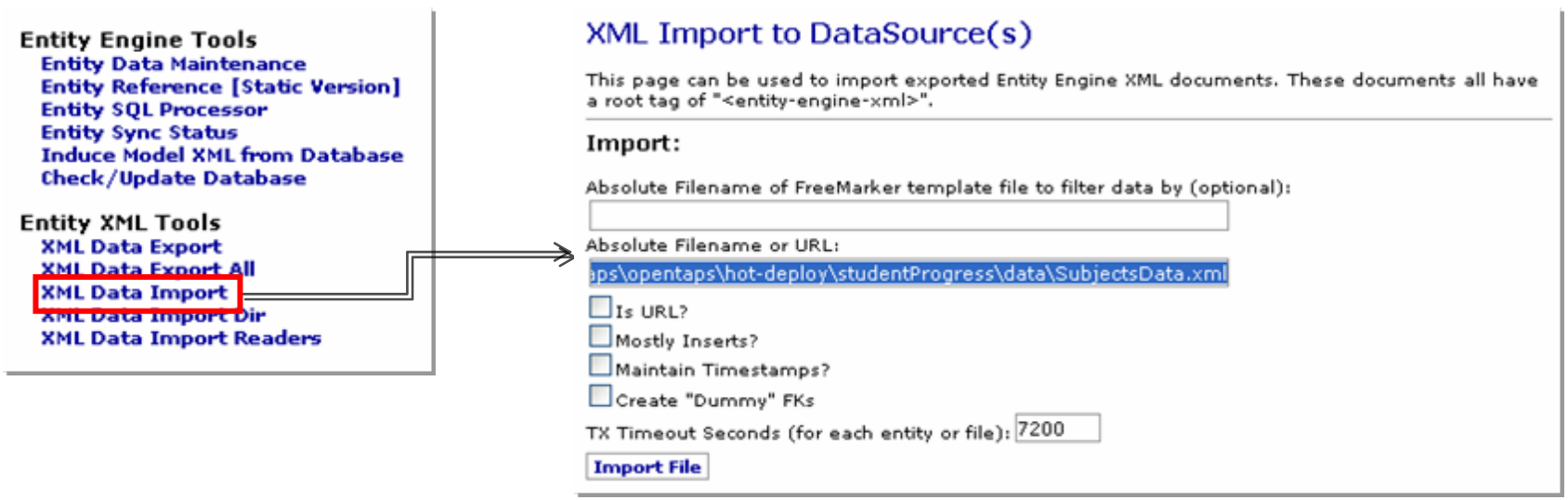
Minilang is simpler, but unknown to must of us. It needs no compile process, it is analysed at application start and is defined especially for common OFBiz application tasks (lookup and store data, work with existing entities and services, …)

In the end: create the web application!

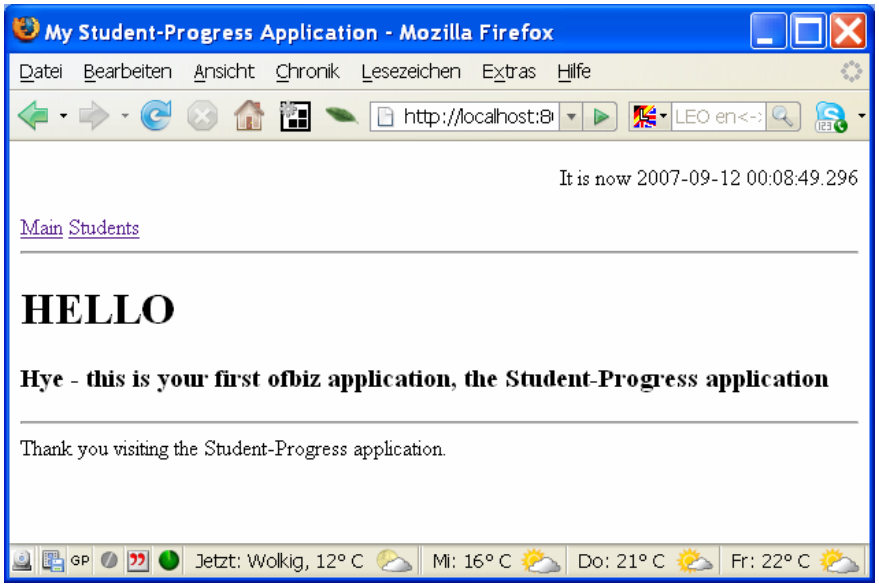
- Differ between usage of screens, forms, ftl files and standard html files!
- See:
  - MyStudentScreens.xml: defines the appearance of each screen in forms of labels, calling forms, …
  - StudentEntryForms.xml: defines parts of a screen, e.g. the insertion part of the student site in forms of fields, drop-down lists, …
  - controller.xml: defines, how the framework has to react on request and view mappings. Its the heart of your application, so to speak.
  - footer.ftl, header.ftl, main.ftl, … define parts of your website

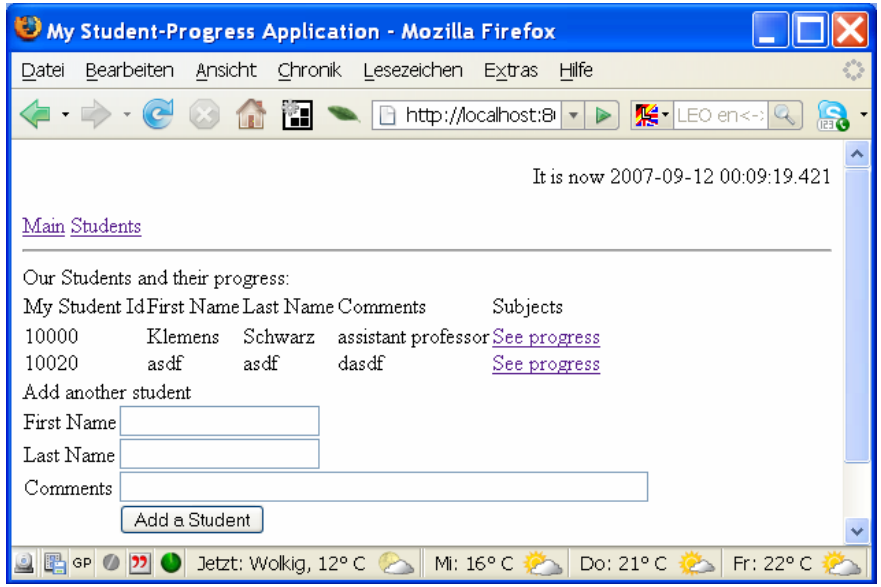
Creating seed/demo data

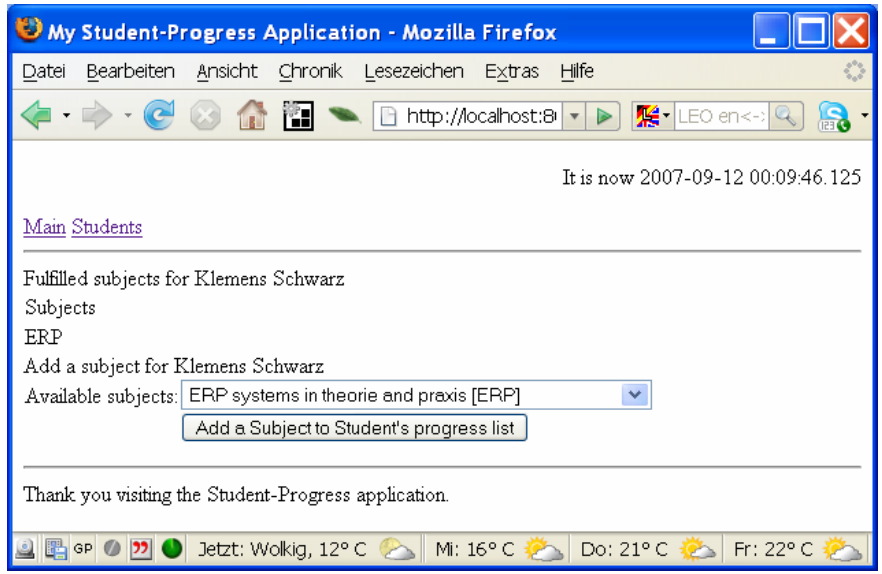
```
<?xml version="1.0" encoding="UTF-8"?>
<entity-engine-xml>
    <!-- Each tag is an entity name.  Data for fields is in attribure-value pairs -->
    <MySubject mySubjectId="ERP" description="ERP systems in theorie and praxis"/>
    <MySubject mySubjectId="PRO" description="Programming basics"/>
    <MySubject mySubjectId="DAT" description="Data Engineering"/>
    <MySubject mySubjectId="SYS" description="System Engineering and Operating Systems"/>
</entity-engine-xml>
```



Results:







Use CSS for better results.

How to work on?

- See http://www.opensourcestrategies.com/ofbiz/tutorials.php for further tutorials, examples, help, …
- Just start ;-)