Netherlands Metabolomics Centre Data Support Platform

Tjeerd Abma, Prasad Gajula, Michael van Vliet, Jeroen Wesbeek (random order)

September 18, 2008
NBIC developers meeting

Netherlands Metabolomics Centre

NMC – Data Support Platform

- **Sharing data between scientists at different locations/institutes**
- **Stimulate using one common standard**
- **Automate data processing**
- **Central repository**
- **Reproducability**
- **Collaboration**
- **Data integrity**

Work approach

- **Four developers sitting at different locations: Leiden, Wageningen, Utrecht, Zeist**
- **Communication via Skype (or other IM-clients)**
- **Every monday come together for a whole day to discuss progress/problems/ideas and to code**
- **Use the NBIC Wiki to plan weekly, to write down information, definitions**
- **Use the Gforge Subversion repository to share the code (commit new code, update, checkout revisions)**
- **Use time-sheets to get a better insight in where time is being spent, useful for future planning**

Gathering information

- **Have interviews with the people in the different labs – what they do, how they do it and what they store. Write it down in a document.**
- **Write down the functional specifications which describes what the user wants (inputs/outputs/properties) in the system. Reach consensus on what is being written down in this document. Everyone should agree on it.**
- **Use a single functional specifications document as a reference and to develop, based on this document, your prototype(s)**

Development process

- **Have iterations in your development process**
- **Build your prototype based on the functional specifications**
- **Evaluate your prototypes once in a while with the people who should be using it. Write down what is missing, should be improved/changed. Try to find consensus what should definitely be in the next prototype**
- **Start developing the next prototype**

Development environment

- **Grails**
  - Open source webframework built on Groovy
- **Groovy**
  - Object-oriented programming language for the Java-platform with features similar to e.g. Python, Ruby
  - Code is dynamically compiled to Java Virtual Machine bytecode and works seamlessly with other Java code and libraries. Most Java code is also syntactically valid Groovy.

Grails

- Previously known as Groovy on Rails
- Started in 2005
- A high-productivity framework following the "coding by convention" paradigm, providing a stand-alone development environment and hiding much of the configuration detail from the developer.
- Current stable release 1.1.1

Grails – features 1/2

- Uses ORM frameworks like Hibernate and Spring to support many databases
- Uses Groovy Server Pages for the view (similar to JSP's)
- Creation of tag libraries to implement custom web page components
- AJAX support
- Easily create a CRUD interface for your webapplication
- Runtime compilation, immediately shows the effects of changes in your code
- Many plugins available on the Grails website
- No XML configuration files needed
- Comes with integrated webserver (Jetty)

Grails – features 2/

- Grails maintained by Spring Source (VMWare)
- Integrates very well with IDE (Netbeans)
- Active community
- Read more at http://www.grails.org

Java vs. Groovy code comparison

```
01 package com.apress.bgg;
02
03 import java.util.List;
04 import java.util.ArrayList;
05 import java.util.Iterator;
06
07 public class Todo {
08   private String name;
09   private String note;
10
11   public Todo() {}
12
13   public Todo(String name, String note) {
14     this.name = name;
15     this.note = note;
16   }
17
18   public String getName() {
19     return name;
20   }
 21
 22   public void setName(String name) {
 23     this.name = name;
 24   }
 25
 26   public String getNote() {
 27     return note;
 28   }
 29
 30   public void setNote(String note) {
 31     this.note = note;
 32   }
 33
 34   public static void main(String[] args) {
 35     List todos = new ArrayList();
 36     todos.add(new Todo("1", "one"));
 37     todos.add(new Todo("2", "two"));
 38     todos.add(new Todo("3","three"));
 39
 40     for(Iterator iter = todos.iterator();iter.hasNext();) {
 41       Todo todo = (Todo)iter.next();
 42       System.out.println(todo.getName() + " " + todo.getNote());
 43     }
 44   }
```

```
01 package com.apress.bgg;
02 public class Todo {
03  String name
04  String note
05 }
06 def todos = [
07  new Todo(name:"1",note:"one"),
08  new Todo(nae:"2", note:"two"),
09  new Todo(name:"3",note:"three")]
10 todos.each {
11  println "${it.name} ${it.note}"
12 }
```

Netherlands Metabolomics Centre

Grails MVC: model

- Domains are the (data)model
  - Contain columns, columntypes, constraints, relationships to other domains
  - See it as a database table you are describing in your (data)model
  - GORM (Grails ORM) will take care of the real mapping of these domains to tables/fields in a database (in memory or persistent)

Grails MVC: view

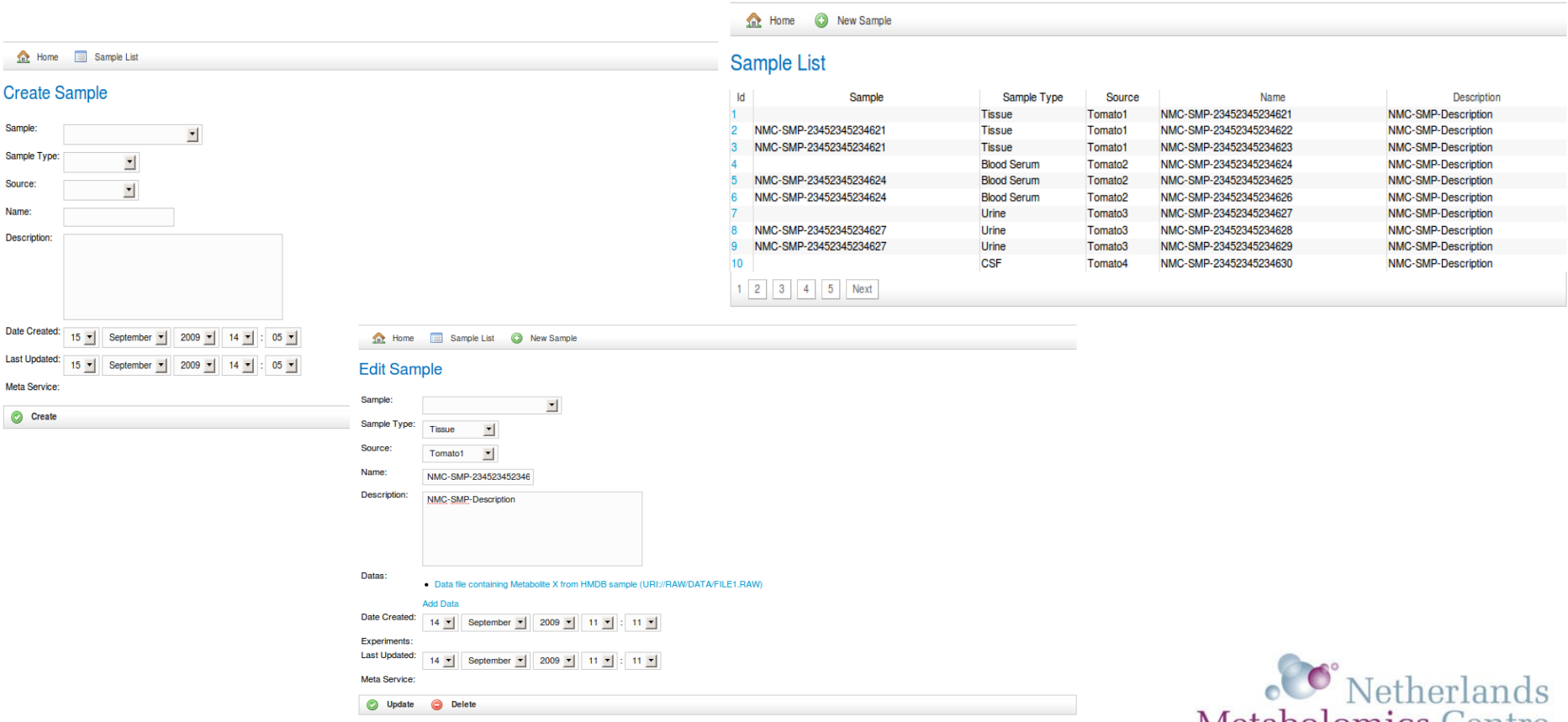
- Views are the presentation in your browser
  - The presentation is written in the so called Groovy Server Pages (also support for JSP's) which is a combination of HTML and specific GSP-tags
  - Views support many tags to quickly inherit components (calendar, dropdownboxes, ajax) into your view
  - The presented view (can) get(s) its information from a model

Grails MVC: controller

- Controllers implement the logic
  - Implement the behaviour of the webpages
  - Do logic to create/update models/objects which can be used by the views

Grails: scaffolding

- When you've defined your domains you can instantly generate a CRUD interface for your application by adding the “scaffolding” parameter to your controllers so you can quickly test your webapplication:



Netherlands Metabolomics Centre

NMC DSP eternal problem

- Continuous problem: what people want to store (fields)/ metadata, keeps changing
- Difficult to keep changing your database fields and keep all your software components synchronized to that structure
- Solution: in-database (metadata) templates

The template structure

- Allows you to create templates which can be put 'on top' of existing tables, so you can actually store metadata/extra fields
- Templates contain a collection of fields (metadata)
- Individual fields can be defined (humidity, lighting, etc.)
- Fields can be grouped
  - Growth environment (group) → humidity, lighting, soil, etc.
- Fields can be of certain types (String, Integer, Date)
- Fields can contain options
  - Color (field) → green, red, blue (options)

In-database templates

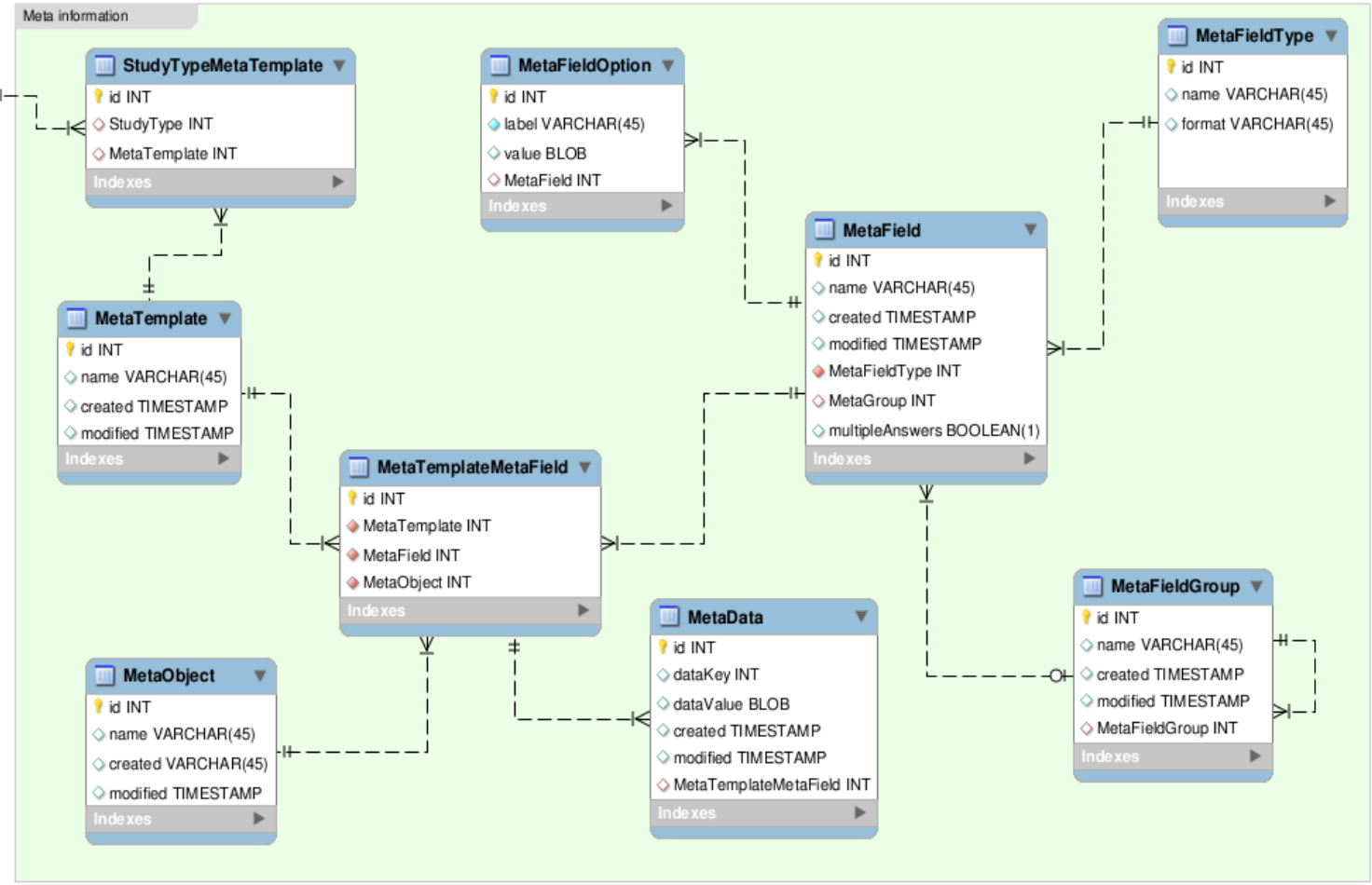
Study

Experiment

Sample

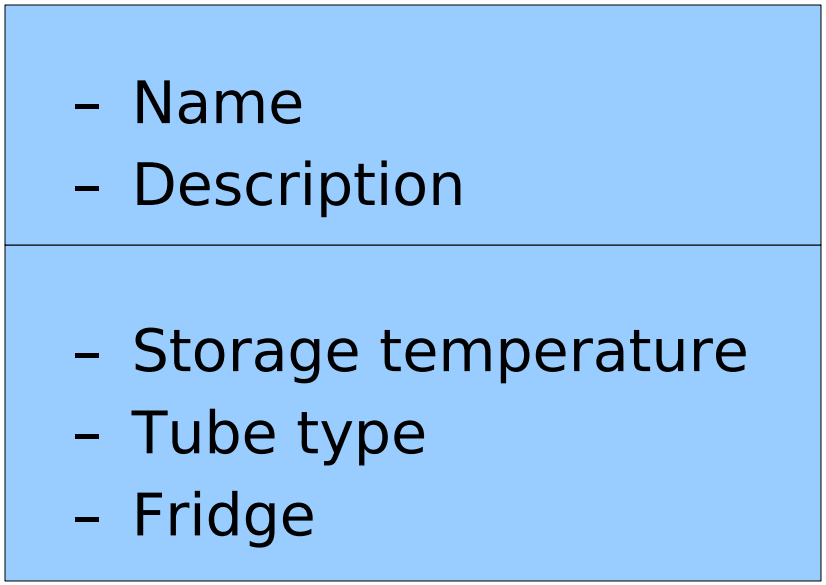
Data

Metadata templates



Template example

- The user will see a form on his screen with fields for a Sample:



'Hard coded' fields in database

Combined with template fields

All above fields will stored/belong to the sample

The template advantages

- Only the administrator (PI for example) can/should create templates/fields
- Fields should/can be based on existing ontologies
- Easily re-use of existing templates (nutritional templates, plant templates, clinical templates), terms, fields and groupings
- Easy way to fit all kinds of meta data/fields
- Support for ISA-TAB format

Template 'disadvantages'

- Avoid the possibility that just everyone can start making templates, keep good control over what templates will be made
- One table is used to store the 'real' metadata, this table will grow fast
- Challenge will be to retrieve back the stored metadata and to write efficient queries for it, this will require more intelligent search queries

Conclusion

- Grails is a flexible environment to rapidly develop our application
- Gather as much as possible information from your 'clients'/ users, write it down, plan, discuss, evaluate
- Templates solve our every changing “what fields should we define for our tables”
- We keep learning

Netherlands Metabolomics Centre

Links

- http://www.grails.org
- https://wiki.nbic.nl/index.php/NMC