

DEVELOPING M2M APPLICATIONS WITH MANGO

Béla Juhász

Bachelor's Thesis
December 2009

Degree Programme in Information Technology
School of Technology



JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES



Author(s) JUHÁSZ, Béla	Type of publication Bachelor's Thesis	Date 11.12.2009
	Pages 40	Language English
	Confidential () Until	Permission for web publication (X)
Title DEVELOPING M2M APPLICATIONS WITH MANGO		
Degree Programme Information Technology		
Tutor(s) PELTOMÄKI, Juha		
Assigned by JAMK University of Applied Sciences		
Abstract <p>Mango is an open source alternative for Machine to Machine software. It enables users to access and control electronic sensors, devices, and machines over multiple protocols simultaneously. However, Mango was designed for the desktop. It relies heavily on JavaScript to render its graphical pages. While rendering, massive amounts of data are being transferred between the Mango server and the browser. Furthermore, because of the continuous polling for new data, it can easily hog the CPU of the computer displaying said data. As a result of these limitations, Mango is hardly usable on a wide range of – mainly older – mobile devices.</p> <p>The commissioners of the project at the JAMK University of Applied Sciences wanted to investigate the possibility of other means of accessing the data collected by Mango. Specifically, the software had to be examined and documented if it was achievable to access the data <i>directly</i> by an external lightweight application written in, for example, PHP.</p> <p>The analysis showed that because of the nature how Mango stores many of its crucial data – by serializing Java objects into BLOB columns – direct access poses challenges. However, with the help of Mango's ability to serialize objects into JSON format, a new path opens to developers aiming for mobile access.</p>		
Keywords Mango, M2M, Machine to Machine, Java, JSON, Spring.		
Miscellaneous		

CONTENTS

1	OBJECTIVE OF THE PROJECT.....	5
2	MANGO M2M SERVER APPLICATION.....	6
2.1	Main concepts in Mango.....	7
2.2	Installation.....	11
2.3	Compilation.....	11
2.4	Software components used.....	13
3	SPRING WEB MVC.....	14
3.1	The MVC architecture	14
3.2	Request processing in Spring MVC	15
3.3	Handler mappings.....	18
3.4	Controllers	18
3.5	Views.....	22
3.6	Ajax with DWR.....	23
3.7	Analysis of a request.....	24
4	DATABASE STRUCTURE IN MANGO.....	30
4.1	Tables visualized.....	30
4.2	Data access layer.....	31
5	AN EXAMPLE MODIFICATION	35
5.1	The task.....	35
5.2	Implementation.....	36
6	CONCLUSION.....	39
	REFERENCES.....	40

FIGURES

FIGURE 1. Graphical representations of sensors

FIGURE 2. An example DataSource with DataPoints

FIGURE 3. Details and actual values of a data point

FIGURE 4. Watch list

FIGURE 5. Graphical view

FIGURE 6. The mobile version of the watch list shown in Figure 4

FIGURE 7. The request processing workflow in Spring Web MVC

FIGURE 8. The most important tables in Mango

FIGURE 9. The details of an HTTP Retriever DataSource

FIGURE 10. An HTTP Image DataSource and its DataPoint

FIGURE 11. The clickable rendered image

TERMINOLOGY

Machine to Machine

Machine to Machine (often abbreviated as M2M) refers to data communications between machines. The term also means the complexity of sensors which produce data, the communications channel which connects the sensors to a server computer

and a software running on the server, having the task to analyse and report the collected data. Sometimes the same software is capable of intervening into the process. (Machine to Machine 2009.)

Open source software

Open source software is a computer software where the source code is provided under a software license that meets the Open Source Definition or it is in the public domain. Open source software therefore can be edited, modified and then redistributed freely, given certain conditions are met (e.g. not placing the software under a more restrictive license, providing the license with the software, etc.) (Open Source Initiative 2009.)

Mango

The open-source Mango software is one – although a crucial – element of a whole M2M system. Its role is to gather and process data, present this raw data in a way which is comprehensible to a human operator who can, in turn, send commands and thus control the process: Mango is an “open source alternative for Machine-to-Machine (M2M) software [...]. Mango is browser-based, Ajax-enabled M2M software that enables users to access and control electronic sensors, devices, and machines over multiple protocols simultaneously.” (Mango, open source M2M 2009.)

MVC

MVC is the acronym for the Model–View–Controller (MVC) architectural pattern used in software engineering. The purpose of this pattern is to simplify the implementation of applications that need to act on user requests and manipulate and display data. The model is the data to be presented to the user. The view part is in charge of rendering the model in a suitable manner. And lastly, the controller is responsible for processing and acting on user requests. It loads and prepares the model, chooses the appropriate view and finally passes the model to the chosen view for rendering. (Machacek, Vukotic, Chakraborty & Ditt 2008, 611.)

Spring's Web MVC framework

Spring is an open source framework, created by Rod Johnson. It was created to address the complexity of enterprise application development. With a more technical description one could describe it as a lightweight dependency injection and aspect-oriented container and framework. For a more in-depth analyses of Spring, refer to Walls 2008. The Spring framework is made up of several well-defined modules built on top of the core container. Such modules are for example the ORM integration module, JMX module, JDBC abstraction and the DAO module. The Spring Web MVC framework is just one of these modules, implementing the MVC architectural pattern. (Ladd, Donald 2006, 7.)

Direct Web Remoting

“DWR, which is short for Direct Web Remoting, is a Java-based Ajax framework that lets you access virtually any server-side Java object through JavaScript. DWR abstracts XMLHttpRequest away so that invoking methods on a server-side Java object is as simple as invoking methods on a client-side JavaScript object.” (Walls 2008, 650.) DWR dynamically converts the server-side Java objects into JavaScript ones that implement the same interface as the original Java objects. The JavaScript libraries are also responsible for sending XMLHttpRequests from the browser to the server and dynamically updating the web page with the data sent back.

Dojo JavaScript toolkit

Dojo is an open source JavaScript framework that simplifies Ajax programming. It is a collection of JavaScript components to assist building JavaScript applications.

1 OBJECTIVE OF THE PROJECT

Originally, Mango was designed for the desktop. It relies heavily on JavaScript to render its graphical pages. While rendering, massive amounts of data are being transferred between the Mango server and the browser. Furthermore, because of the continuous polling for new data, it can easily hog the CPU of the computer displaying said data. As a result of these limitations, Mango is hardly usable on a wide range of – mainly older – mobile devices. Mobile devices tend to be slow, memory is always scarce and the size of the display is barely enough to show the graphical views originally created for a high-resolution monitor of a desktop computer.

Fortunately, with the arrival of the latest version of Mango (1.7.0), a very simple mobile `WatchList` has appeared in the program. It is a much simplified version of the original `WatchList` page and was written with mobile devices in mind. However, it lacks many of the functionalities needed by the commissioners of the project at the JAMK University of Applied Sciences, therefore there was a need that someone implemented those requirements.

Among these requirements is

- the ability to attach simple event detectors to point values with the accompanying event handlers
- a mobile `WatchList` specifically written to a certain type of mobile phone
- mass-creation of `DataSources`
- the ability of uploading pictures (to make `WatchLists` easier to be identified)
- the possibility of directly accessing the data stored in Mango from – for example – PHP
- using the data in an external application after having it exported into JSON

format.

For all this to be completed, one has to first understand and document the system. This thesis aims to be the first step towards that goal. The task of the author of this thesis was to discover how Mango works beneath the covers, to document it and, if it is possible, to make some minor modifications to the system.

2 MANGO M2M SERVER APPLICATION

As stated before, Mango is one important element of a complete M2M system. It provides a database to store the collected data, a Human-Machine Interface to provide graphs, diagnostic data, and management information. Mango can present the information to the operating personnel in an easy to use graphical interface, in the form of a mimic diagram. For example, a picture of a thermometer can show the operator the temperature of the room where one of the sensors is located.

2.949524009670901

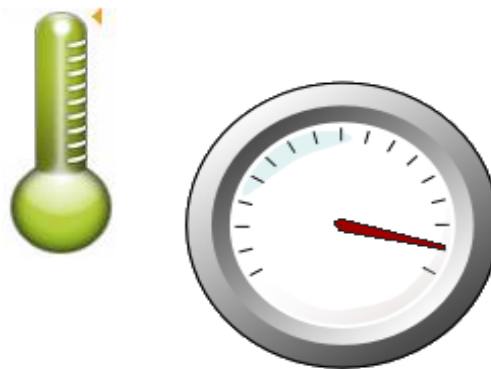


FIGURE 1. Graphical representations of sensors

Furthermore, it enables the administrator to set up alarms on specific events, to log these events, and to react to these events in an automated fashion. The system is designed for multiple users: it has a sophisticated user management system, where one can finely tune what ordinary users can see and do.

Mango is a highly-regarded software in the world of M2M automation. Being open

source, the program is not just free, but is also freely available for any modification.

It must be admitted, that it also has some disadvantages, too. The code got very complex over the years, it seems to be developed by only one person and the user community is everything but large. Still, currently it is one of the best free solution to be found on the market.

2.1 Main concepts in Mango

Though the task is to describe Mango from a developer's point of view (and not from a user's), the main concepts of the system need to be presented, because they will be referred to on several occasions.

Collecting data begins with setting up a “data source”. Different kind of data sources are available, based on their underlying protocol for gathering data. Examples for protocols are Modbus IP, Modbus Serial, 1-wire, SNMP. Mango can also receive data through HTTP: it can download an image, and it is able to parse HTML, so with the help of a well-formed regular expression, one can easily extract valuable information (e.g. temperature data) from any web page (website of a weather station). Related source files are `com.serotonin.mango.rt.dataSource.DataSourceRT` and `com.serotonin.mango.vo.dataSource.DataSourceVO`. `DataSourceVO` represents the configuration of a `DataSourceRT`, and it is the `DataSourceVO` whose parameters can be set at the `data_source_edit.shtm`. `DataSourceRT` is only an interface for the specific data sources implemented under the `/src/com/serotonin/mango/rt/dataSource` directory.

The screenshot shows the Mango Full Service M2M web interface. At the top left is the Mango logo with the tagline "Full Service M2M". At the top right is an "Information" icon. Below the header is a navigation bar with various icons. The main content area is titled "Data sources" and features a dropdown menu set to "BACnet I/P". Below this is a table with columns for Name, Type, Connection, and Status. The table lists three data sources: "http image name", "my-virtual-data-source", and "weather". Each data source has a sub-table of data points with columns for Point name, Description, and Status. The "http image name" source has a point named "image point" with a description "Image / http://a09b.keltti.jyu.fi:8080/mango17/uploads/55.jpg". The "my-virtual-data-source" has a point named "my-virtual-data-point" with a description "Numeric / Random". The "weather" source has two points: "conditions" with a description "Alphanumeric / Regex: Conditions: (.*) \\" and "humidity" with a description "Numeric / Regex: Humidity: (.*)\D". At the bottom of the page, it says "Page 1 of 1 (1 - 3 of 3 rows)" and "©2006-2009 Serotonin Software Technologies Inc., all rights reserved".

Name	Type	Connection	Status
http image name	HTTP Image	image stream every 1 minutes(s)	
Point name Description Status			
image point	Image / http://a09b.keltti.jyu.fi:8080/mango17/uploads/55.jpg		
my-virtual-data-source	Virtual Data Source	30 second(s)	
Point name Description Status			
my-virtual-data-point	Numeric / Random		
weather	HTTP Retriever	http://rss.wunderground.com/au ...	
Point name Description Status			
conditions	Alphanumeric / Regex: Conditions: (.*) \\"		
humidity	Numeric / Regex: Humidity: (.*)\D		

Page 1 of 1 (1 - 3 of 3 rows) 1

©2006-2009 Serotonin Software Technologies Inc., all rights reserved

FIGURE 2. An example DataSource with DataPoints

After setting up a data source, one must define so-called “data points”. One data source can have many different data points. A physical measurement device (a data source, that is) can, for example, supply data about the temperature, the humidity and the brightness of a room. The latter three objects are the data points. It is crucial not to confuse data points with their actual values: a data point is simply a concept (i.e. humidity), while its value will be 10%. Data points thus have data point values. Data points do not change, only their values do. Interesting files are the following:

`com.serotonin.mango.vo.DataPointVO,`

`com.serotonin.mango.rt.dataImage.DataPointRT` and

`com.serotonin.mango.db.dao.DataPointDao`. The actual values are represented

by the `com.serotonin.mango.rt.dataImage.PointValueTime` class, and are handled by the following classes:

`com.serotonin.mango.rt.dataImage.PointValueFacade` and

`com.serotonin.mango.db.dao.PointValueDao`.

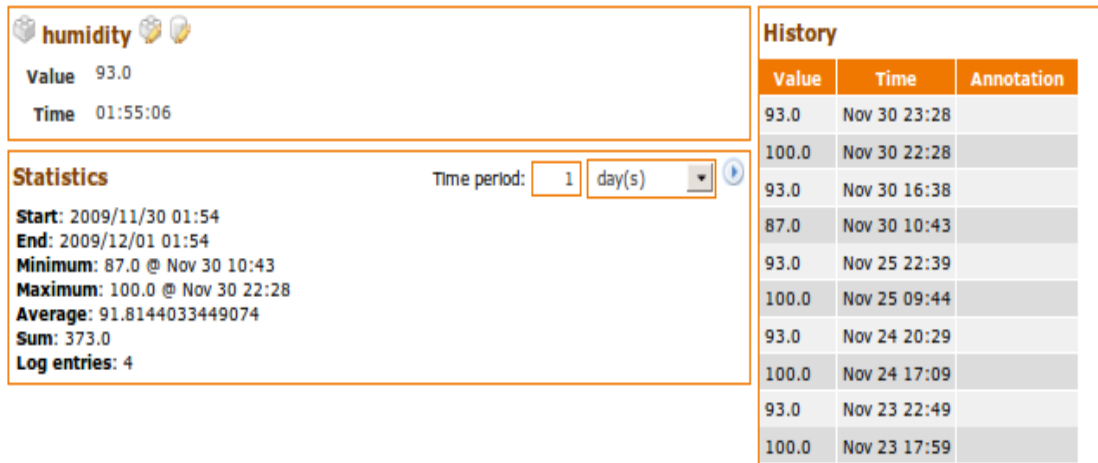


FIGURE 3. Details and actual values of a data point

One can examine and follow these data points (and their values) by using “watch lists” or “graphical views”. Watch lists are a tabular set of points that one wishes to view all at once. Point values and value times update automatically, there is no need to refresh the page.

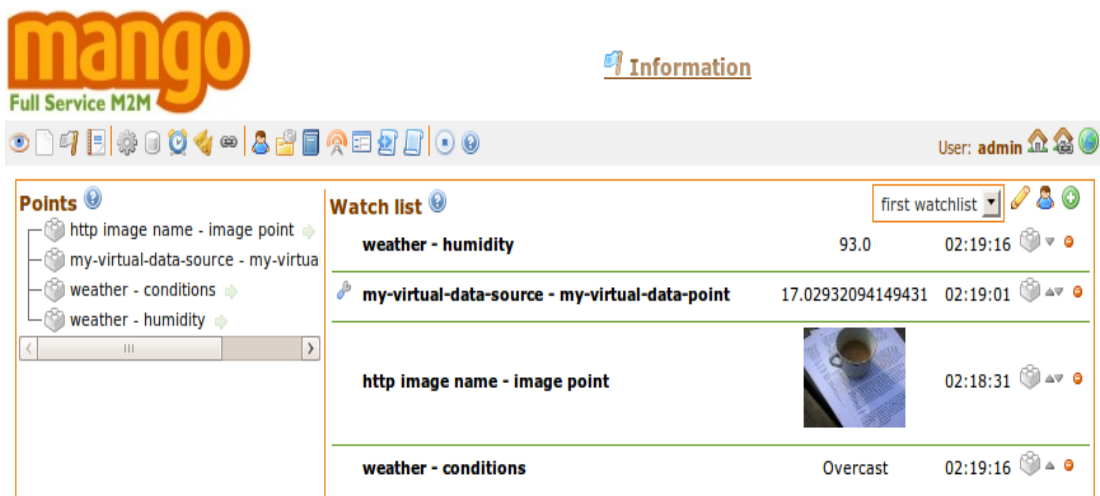



FIGURE 4. Watch list

A graphical view is similar to a watch list, but instead provides a graphical representation of selected points. The representation of each point in one's view depends upon that point's configuration. Points can be rendered as simply a current value (like in a watch list), or an arbitrary icon set can be used that visually represents the point's value (e.g. a thermometer showing the temperature). Graphical views are visually more pleasing to the eye.



FIGURE 5. Graphical view

For example, graphical views may optionally be presented upon background images. In this thesis the emphasis is on watch lists, so for now, the interesting classes are: `com.serotonin.mango.vo.WatchList`, `com.serotonin.mango.web.dwr.WatchListDwr` and `com.serotonin.mango.db.dao.WatchListDao`. With the 1.7.0 release of Mango, a new concept, the so-called “mobile watch lists”, has appeared. These are simplified, lightweight versions of the original heavyweight watch lists. Normal watch lists use massive amount of JavaScript, they are slow or sometimes even unable to load on mobile devices, that is why something lighter was needed. `com.serotonin.mango.web.mvc.controller.MobileWatchListController` and `com.serotonin.mango.web.mvc.controller.MobileWatchListState` are the most important classes here.

weather - humidity	93.0	02:20:06
my-virtual-data-source - my-virtual-data-point	8.197638351363407	02:20:01
http image name - image point		02:19:31
weather - conditions	Overcast	02:20:06

Watch list: [Reload](#) [Logout](#)

©2006-2009 Serotonin Software Technologies Inc., all rights reserved

FIGURE 6. The mobile version of the watch list shown in Figure 4

Mango provides a robust and highly configurable event management system, where users can programmatically react to specific events. One can define and attach an “event detector” to a data point (for example, the temperature of the room exceeded a certain limit), after which the “event handling” system kicks in and alerts those users (by email), who are subscribed for this type of event.

2.2 Installation

Installation is fairly straightforward, although not without problems. Instructions can be found on the Mango homepage (<http://mango.serotoninsoftware.com/download.jsp>), solutions for common installation problems are in the Forum (<http://mango.serotoninsoftware.com/forum>).

2.3 Compilation

The main objective is to modify the Mango source code in a way so that it is usable on mobile devices. Mango, fortunately being an open source application, comes with its own source code. At the time of writing this thesis, the source is distributed as a zip package, although there are plans to host it in a repository of a revision control system. The current maintainer uses Eclipse as his main development tool, but not wanting to force the reader to use this specific IDE, the author of this thesis chose to use only a simple text-editor and the command-line armed with ant. To be able to

compile it, the following modifications had to be made to the source code:

- In the file `$MANGO_HOME/build.properties`:

- `tomcat.home`

This is the directory where Tomcat is residing, e.g.
`/home/user/opt/tomcat6`

- `tomcat.manager.username`

- `tomcat.manager.password`

- `tomcat.appdir`

The name of the directory where the compiled Mango will be deployed to.
In this example it is `mango17`.

- `tomcat.apppath`

The absolute path of the directory (relative from the `webapps` directory of Tomcat) where the compiled Mango will be deployed to. In this example it is `/mango17`.

- `db.url`

- Download `RXTXcomm.jar` from the Mango homepage and put it in `$MANGO_HOME/lib`

- Download `axis-ant.jar` (for example from the `apache.org` website or from your Eclipse distribution) and put it in `$MANGO_HOME/lib`

- Delete the

`$MANGO_HOME/src/com/serotonin/mango/util/test/OpCTest.java` file. This is only a test file, which unfortunately did not compile, and fixing it would take too much time. It is safe to delete.

- Copy the `$MANGO_HOME/build.properties` to `$MANGO_HOME/war/WEB-INF/classes/env.properties`, modify its contents to look similar to this, then adjust the `db.url` property according to your setup.

```
db.type=derby
db.url=~/.mangoDB
db.username=
db.password=

db.pool.maxActive=10

db.pool.maxIdle=10

convert.db.type=

convert.db.url=
convert.db.username=

convert.db.password=

grove.url=http://mango.serotoninsoftware.com/➡

servlet
```

Remark: `$MANGO_HOME` is the directory where the source code has been extracted to. After all these changes have been made, issuing an `ant` and `ant reload` command should run without problems.

2.4 Software components used

The newest available release of Mango is at version 1.7.0. Since there is no public source code repository, there are no other (unstable) releases either.

At the time of writing this thesis, the newest release of Mango uses the following software components and frameworks:

- Spring's Web MVC framework, version 2.5.3, <http://www.springsource.org>

- Direct Web Remoting, version 2.0.1, <http://directwebremoting.org/dwr/>
- Dojo JavaScript toolkit, version 0.4.2, <http://www.dojotoolkit.org>, <http://dev.aol.com/dojo>. According to the author of Mango, the reason for using such an outdated version is that updating to a newer one would be a hard and tedious task. Moreover, since the current version functions properly, there is no urgent need for updating it. Furthermore, the newer versions of Dojo are not on par with features of this old version, according to an insightful user's comment (Updating Dojo 2009).

3 SPRING WEB MVC

3.1 *The MVC architecture*

MVC stands for Model-View-Controller. It is an architectural pattern, and it enables the separation of business logic from data and presentation. In this way the different layers of the software can be developed and tested independently.

The model represents the data which should be presented to the user. In the case of Spring, this is a `Map` object, “which contains bean names and corresponding objects” (Spring Web MVC framework 2009).

The view renders the model in appropriate formatting, as in this case, into HTML. Spring supports many kind of rendering technology, for example JSP, Velocity and Tiles. Spring's view resolution (the mechanism to find the view to be rendered) is extremely flexible and highly configurable.

The controller has the task to process user requests and to respond to them. It creates the model object, then passes it to the view which finally presents it in a suitable form. Spring offers a huge variety of controllers (plain, command, form, wizard, multi-action or a custom one), suitable for everyone's need.

The three parts of the MVC architecture is orchestrated by a servlet called

DispatcherServlet that dispatches requests to controllers and view resolvers.

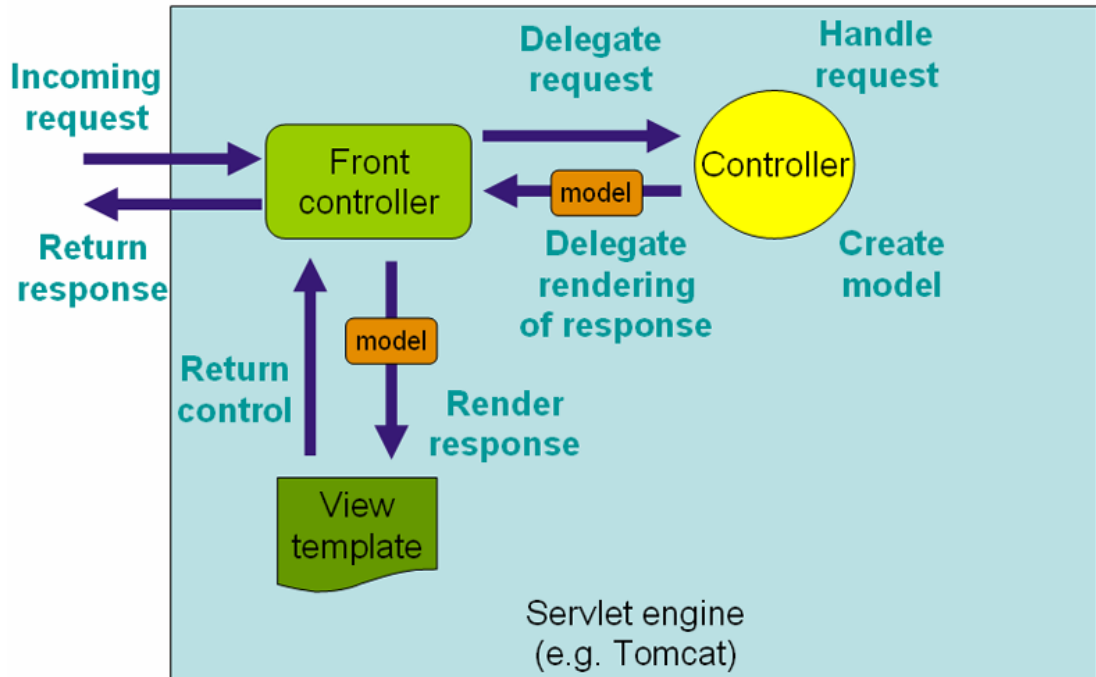


FIGURE 7. The request processing workflow in Spring Web MVC (Spring Web MVC framework 2009)

3.2 Request processing in Spring MVC

3.2.1 Choosing the right controller with the help of the handler mappings

When the user's request leaves the browser, it first arrives at the DispatcherServlet. DispatcherServlet is the implementation of the Front Controller design pattern, found in many web frameworks. DispatcherServlet consults one or more handler mappings to choose the most appropriate controller, then it simply forwards the request to that chosen controller.

In the original Mango distribution, DispatcherServlet uses the handler mapping called SimpleUrlHandlerMapping to find the right controller to handle the request. With SimpleUrlHandlerMapping, one maps a URL pattern directly to a controller:

```

<!--
  URL mappings to controllers
-->
<bean id="publicUrlMappings" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <!-- All user URLs -->
      <prop key="/compound_events.shtm">compoundEventsController</prop>
      <prop key="/data_point_details.shtm">dataPointDetailsController</prop>
      <prop key="/data_point_edit.shtm">dataPointEditController</prop>
      <prop key="/data_source_edit.shtm">dataSourceEditController</prop>
      <prop key="/data_sources.shtm">dataSourceListController</prop>
      <prop key="/event_handlers.shtm">eventHandlersController</prop>
      <prop key="/events.shtm">eventsController</prop>
      <prop key="/login.htm">loginController</prop>
      <prop key="/logout.htm">logoutController</prop>
      <prop key="/mailing_lists.shtm">mailingListsController</prop>
      <prop key="/point_hierarchy.shtm">pointHierarchyController</prop>
      <prop key="/point_links.shtm">pointLinksController</prop>
      <prop key="/public_view.htm">publicViewController</prop>
      <prop key="/reports.shtm">reportsController</prop>
      <prop key="/reportChart.shtm">reportChartController</prop>
      <prop key="/scheduled_events.shtm">scheduledEventsController</prop>
      <prop key="/sql.shtm">sqlController</prop>
      <prop key="/system_settings.shtm">systemSettingsController</prop>
      <prop key="/users.shtm">usersController</prop>
      <prop key="/views.shtm">viewsController</prop>
      <prop key="/view_edit.shtm">viewEditController</prop>
      <prop key="/watch_list.shtm">watchListController</prop>
      <prop key="/webcam_live_feed.htm">webcamLiveFeedController</prop>

      <!-- Mobile user URLs -->
      <prop key="/mobile_login.htm">mobileLoginController</prop>
      <prop key="/mobile_logout.htm">mobileLogoutController</prop>
      <prop key="/mobile_watch_list.shtm">mobileWatchListController</prop>
    </props>
  </property>
</bean>

```

EXAMPLE 1. war/WEB-INF/springDispatcher-servlet.xml

3.2.2 Controller builds a ModelAndView object

The responsibility of the chosen controller is to return a ModelAndView object. The ModelAndView object consists of two other objects: a Map instance with name-value pairs holding the model data and a logical name of a view component.

```

List books = getBooks();
User customer = getCustomer();
Map model = new Map();
model.add("books", books);
model.add("user", customer);

ModelAndView mav = new ModelAndView();
mav.add("home", model);

```

The logical view name does not directly reference a JSP page. Later, the view resolver's task will be to decide which specific JSP to render.

This ModelAndView object is then returned to the DispatcherServlet, which in turn asks a view resolver.

3.2.3 View resolver decides which actual JSP to render

Just like handler mappings, one can find several view resolvers, too, each with a different strategy to resolve a logical view name into an actual one. Mango uses the InternalResourceViewResolver.

```
<!-- View resolver -->
<bean id="viewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass">
    <value>org.springframework.web.servlet.view.JstlView</value>
  </property>
  <property name="prefix">
    <value>/WEB-INF/jsp/</value>
  </property>
  <property name="suffix">
    <value>.jsp</value>
  </property>
</bean>
```

EXAMPLE 2. war/WEB-INF/springDispatcher-servlet.xml

This resolver does nothing but glues together the prefix property, the logical view name and the suffix property.

```
<bean id="dataSourceEditController"
  class="com.serotonin.mango.web.mvc.controller.DataSourceEditController">
  <property name="viewName"><value>dataSourceEdit</value></property>
</bean>
```

EXAMPLE 3. war/WEB-INF/springDispatcher-servlet.xml

Thus a logical view with the value dataSourceEdit becomes /WEB-INF/jsp/dataSourceEdit.jsp.

3.2.4 The page gets rendered

DispatcherServlet sends the model data to the view implementation (usually a JSP file), and the view layer will render a page to the user using the given data.

3.3 *Handler mappings*

“Using a handler mapping you can map incoming web requests to appropriate handlers. There are some handler mappings you can use out of the box, for example, the `SimpleUrlHandlerMapping` or the `BeanNameUrlHandlerMapping`” (Spring Web MVC framework 2009). Mango by default uses `SimpleUrlHandlerMapping`: “This mapping is configurable in the application context and has Ant-style path matching capabilities.” (Op. cit.) For an example for this type of handler mapping see Example 1.

3.4 *Controllers*

Spring provides a wide variety of controllers in a somewhat puzzling controller hierarchy. Fortunately, Mango only uses a few of them. What follows now, is a short description of each controllers used in Mango, along with an example taken from the sources.

3.4.1 **AbstractController**

This type of controller is used when the controller requires “little more functionality than is afforded by basic Java servlets” (Walls 2008, 508). In line with the Open-Closed Principle (Martin 1996), the `handleRequest()` method is marked as `final`. The entry point for extending the class is the `handleRequestInternal()` method. `AbstractController` provides a few useful functionalities (e.g. enforcing HTTP methods, cache header management, checking for the existence of a session, etc.), but these are not used by the controllers found in Mango. `AbstractController` is useful, when it is enough to retrieve parameters from the `HttpServletRequest` itself, without binding those parameters automatically to a command object. `ParameterizableViewController` is similar to `AbstractController`, “except

for the fact that you can specify the view name that it will return in the web application context (and thus remove the need to hard-code the viewname in the Java class)” (Spring Web MVC framework 2009).

For example,

`com.serotonin.mango.web.mvc.controller.LogoutController.java` checks if the user is logged in,

```
User user = Common.getUser(request);
```

and then acts accordingly:

```
request.getSession().invalidate();
```

3.4.2 AbstractCommandController

Though `AbstractCommandController` is not used in Mango, it is a fairly important type of controller, because `CommandControllers` (such as `AbstractCommandController` and `BaseCommandController`) automatically bind request parameters to a so-called command object. A command object is nothing but a `JavaBean`. Usually, it also happens to be a domain object (e.g. `User`, `Article`), this way one does not have to implement the same class twice, when he or she wants to populate the said object directly from requests.

In `AbstractCommandController`, the extension point to place custom logic is the `handle()` method. But before the `handle()` method is called, any parameters found in the request object are matched to properties in the command object. If a match is found, then the value of the parameter is bound to the object's property.

Request parameters can come from URL parameters (if it was a GET request) or fields from a form (POST request). `AbstractCommandController` is able to handle all these, but there are classes more suitable for this task.

3.4.3 SimpleFormController

Together with `AbstractFormController`, `SimpleFormController` provides an easy way for processing web forms. `FormControllers` display a form when they receive a GET request. Upon submitting that form, the controller processes the input

(i.e. binds the input variables to a command object) and either returns a success page or redisplay the form (if any errors occurred).

Request and form binding

`CommandControllers` and `FormControllers` all have a command object associated with them, whose properties are bound automatically, in a two-way fashion.

When a form is submitted (that means a POST request) or a GET request arrives with additional parameters, the properties of the command object are populated with the values of the corresponding request parameters, and then the command object is processed by the controller.

When rendering a form, a similar binding happens, only in the other way: the values of the input fields of the form are filled in by the values of the command object, thus, when a form has to be re-rendered because of an error, the fields will be re-filled by their previous values entered by the user.

Conversion

The Servlet API returns request and form parameters only as Strings. To be able to bind non-String parameters to a command bean, Spring uses `PropertyEditors`, which happily convert String parameters into nearly any other class.

Binding nested properties

If there happens to be an input field with the name `"user.firstName"` and with the value `"Adam"`, then the `org.springframework.validation.DataBinder` class translates that into a `getUser().setFirstName("Adam")` method call on the command object. With the same mechanism, `DataBinder` supports binding to Collections, Arrays, Maps, Sets, and with the help of `PropertyEditors`, even non-String properties can be parsed into Java objects (e.g. `java.io.File`, `java.net.URL`, `Boolean`, etc). `DataBinder` is also able to perform basic validation, but for more complex validation, one has to use Spring's Validation framework.

Validation

After the input fields were bound to the command bean, an optional validation phase follows. By implementing the `org.springframework.validation.Validator` interface, one can validate the fields of a given object. Since Mango validates input in a different way (by overriding the `onBindAndValidate()` method), examining validation with the `Validator` interface is skipped.

Additional information for a form

Additional information (for example, a list of all the countries in the world, ready to be fed to an HTML select-box) for the form to be displayed can be provided by overriding the `referenceData()` method, like in the `com.serotonin.mango.web.mvc.controller.ViewEditController` or `com.serotonin.mango.web.mvc.controller.DataPointEditController`. This method returns a `Map` object similar to the `Model` part of the `ModelAndView` object.

Life cycle of a form

`SimpleFormController` is responsible for handling the entire life cycle of an HTML form. It displays the form when it receives a GET request, it calls the `onSubmit()` method if the request is of the type POST and redisplay the form if it contains errors. On success, it redirects to the page defined by the `successView` property in the context configuration file (in Mango's case, this is the `war/WEB-INF/springDispatcher-servlet.xml` file). The form to be displayed is defined by the `formView` property in the same XML file. The command object can also be set declaratively in the same XML by using the `commandName` and `commandClass` properties.

```

<bean id="loginController"
  class="com.serotonin.mango.web.mvc.controller.LoginController">
  <property name="commandName"><value>login</value></property>
  <property name="commandClass">
    <value>com.serotonin.mango.web.mvc.form.LoginForm</value>
  </property>
  <property name="formView"><value>login</value></property>
  <property name="successUrl">
    <value>watch_list.shtm</value>
  </property>
  <property name="newUserUrl"><value>help.shtm</value></property>
  <property name="bindOnNewForm"><value>true</value></property>
</bean>

```

EXAMPLE 4. war/WEB-INF/springDispatcher-servlet.xml

Thus, the `LoginController` (which extends the `SimpleFormController`) first displays that JSP page which has the logical name `login` defined by the `formView` property. This is the same page which gets displayed when any errors are encountered. Upon a successful submit, it displays a JSP page which gets looked up by decoding the logical name set by the `successUrl` property. A view resolver (clarified later) will be used to locate the actual JSP file.

To avoid the dreaded double-submit bug (when the user reloads the form for a second time) it is best to use the Redirect After Submit pattern. In Spring, just pass the `RedirectView` object as the View object to be rendered:

```

return new ModelAndView(
    new RedirectView(successUrl));

```

In this way, when the user reloads the page, he or she only refreshes the success page and not the original form. Mango has a special controller called `SimpleFormRedirectController` which just happens to accomplish this pattern. It is basically a `SimpleFormController`, the only difference is that the view name is set not through the `springDispatcher-servlet.xml` file, but by a query string passed to the `getSuccessRedirectView()` method.

3.5 Views

Spring controllers must return a `ModelAndView` object, which contains either a `View` object or a logical view name, which in turn gets resolved into a `View` bean by the

help of a view resolver. There are several view resolving strategies implemented by Spring's view resolvers. Mango uses the `InternalResourceViewResolver` one (defined in the `springDispatcher-servlet.xml`), which only strings together the value of the `prefix` property, the logical name and the value of the `suffix` property.

```
<!-- View resolver -->
<bean id="viewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass">
    <value>org.springframework.web.servlet.view.JstlView</value>
  </property>
  <property name="prefix">
    <value>/WEB-INF/jsp/</value>
  </property>
  <property name="suffix">
    <value>.jsp</value>
  </property>
</bean>
```

EXAMPLE 5. `war/WEB-INF/springDispatcher-servlet.xml`

Thus after `DataSourceEditController` calls `getViewName()`,

```
return new ModelAndView(getViewName(), model);
```

the `ModelAndView` object, which gets created, will contain the logical view name `dataSourceEdit`,

```
<bean id="dataSourceEditController"
  class="com.serotonin.mango.web.mvc.controller.DataSourceEditController">
  <property name="viewName"><value>dataSourceEdit</value></property>
</bean>
```

EXAMPLE 6. `war/WEB-INF/springDispatcher-servlet.xml`

which, in the end, the view resolver translates into `/WEB-`

`INF/jsp/dataSourceEdit.jsp`. `InternalResourceViewResolver` then “simply dispatches the request to the JSP to perform the actual rendering” (Walls 2008, 536).

The model data (which is part of the `ModelAndView` object) is passed to the JSP, in which the input fields are bound to the properties of the command object, respectively. Data binding occurs as it was described in **Request and form binding**.

3.6 Ajax with DWR

The file `war/WEB-INF/dwr.xml` contains the list of classes to be exposed to the

client-side JavaScripts. The `DWRServlet`, configured in `web.xml`, is responsible for creating the JavaScript versions of the original Java classes. This code creation happens on-the-fly. Every file under the `/dwr` path is served by the `DWRServlet`, thus if a request arrives to the server with the path `/dwr/interface/MiscDwr.js`, then the servlet dynamically creates the `MiscDwr.js` file, which contains all the exposed methods of the `MiscDwr` Java class. Then a `MiscDwr.doLongPoll(mango.header.longPoll.pollCB)` JavaScript method call is routed to the original `MiscDwr` Java object. The result of the call is a `Map<String, Object>` object, which is serialized to JSON format and sent back to the DWR script running in the browser. Because of the nature of Ajax, the response is not necessarily delivered immediately, but, as the 'A' in Ajax suggests, asynchronously. That is why there is a need for a callback function. In this example, the callback function is called `mango.header.longPoll.pollCB()`, and it is the only argument of the `MiscDwr.doLongPoll()` call. And exactly this is the callback function which receives the result containing the `Map<String, Object>` object, which, in the end, gets injected into the HTML code of the page.

3.7 Analysis of a request

What happens from the time the user issues the `http://localhost:8080/mango/public_view.htm?viewId=1` request until the page gets fully rendered in his or her browser?

The request first arrives at the Mango application running in a Tomcat servlet.

In the file `war/WEB-INF/web.xml` there is a servlet-mapping with the `*.htm` pattern,

```
<!--  
  Servlet definitions.  
-->  
<servlet>  
  <servlet-name>springDispatcher</servlet-name>  
  <servlet-class>  
    org.springframework.web.servlet.DispatcherServlet  
  </servlet-class>  
  <load-on-startup>1</load-on-startup>  
</servlet>  
  
<!--  
  Servlet mappings.  
-->  
<servlet-mapping>  
  <servlet-name>springDispatcher</servlet-name>  
  <url-pattern>*.htm</url-pattern>  
</servlet-mapping>
```

EXAMPLE 7. war/WEB-INF/web.xml

so the request is routed to

org.springframework.web.servlet.DispatcherServlet.

DispatcherServlet reads in the war/WEB-INF/springDispatcher-servlet.xml file and searches for handler mappings.

Upon finding the SimpleUrlHandlerMapping handler mapping, it searches through its mapping directives,

```

<!--
  URL mappings to controllers
-->
<bean id="publicUrlMappings" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <!-- All user URLs -->
      <prop key="/compound_events.shtm">compoundEventsController</prop>
      <prop key="/data_point_details.shtm">dataPointDetailsController</prop>
      <prop key="/data_point_edit.shtm">dataPointEditController</prop>
      <prop key="/data_source_edit.shtm">dataSourceEditController</prop>
      <prop key="/data_sources.shtm">dataSourceListController</prop>
      <prop key="/event_handlers.shtm">eventHandlersController</prop>
      <prop key="/events.shtm">eventsController</prop>
      <prop key="/login.htm">loginController</prop>
      <prop key="/logout.htm">logoutController</prop>
      <prop key="/mailing_lists.shtm">mailingListsController</prop>
      <prop key="/point_hierarchy.shtm">pointHierarchyController</prop>
      <prop key="/point_links.shtm">pointLinksController</prop>
      <prop key="/public_view.htm">publicViewController</prop>
      <prop key="/reports.shtm">reportsController</prop>
      <prop key="/reportChart.shtm">reportChartController</prop>
      <prop key="/scheduled_events.shtm">scheduledEventsController</prop>
      <prop key="/sql.shtm">sqlController</prop>
      <prop key="/system_settings.shtm">systemSettingsController</prop>
      <prop key="/users.shtm">usersController</prop>
      <prop key="/views.shtm">viewsController</prop>
      <prop key="/view_edit.shtm">viewEditController</prop>
      <prop key="/watch_list.shtm">watchListController</prop>
      <prop key="/webcam_live_feed.htm">webcamLiveFeedController</prop>

      <!-- Mobile user URLs -->
      <prop key="/mobile_login.htm">mobileLoginController</prop>
      <prop key="/mobile_logout.htm">mobileLogoutController</prop>
      <prop key="/mobile_watch_list.shtm">mobileWatchListController</prop>
    </props>
  </property>
</bean>

```

EXAMPLE 8. war/WEB-INF/springDispatcher-servlet.xml

only to find out that the `/public_view.htm` page is served by the controller named `publicViewController`.

`PublicViewController` is a subclass of the `ParameterizableViewController` class, which is one of the simplest form of the controller types provided by Spring. It does nothing, just returns a view, named by the configuration property `viewName`. This configuration property is set in the `springDispatcher-servlet.xml` file and, in this case, its value is `publicView`.

The entry point for extending the `ParameterizableViewController` class is the `handleRequestInternal()` method, that is why this method is the one which is overridden in the `com.serotonin.mango.web.mvc.controller.PublicViewController` class.

PublicViewController extracts the request parameter called `viewId` from the `HttpServletRequest` object and, with the help of a `ViewDao` object, finds and creates a `view` object specified by the `viewId` parameter. The `view` was previously saved into the database, now its details are read from there.

After checking that the `view` and its components are accessible even for an anonymous user, the `view` is added to the model, so it will be available to the JSP page.

```
Map<String, Object> model = ↪
    new HashMap<String, Object>();
model.put("view", view);
```

The `getViewName()` call returns the value of the `publicView` configuration property set in the `springDispatcher-servlet.xml` file. The controller finishes by returning a `ModelAndView` object.

```
return new ModelAndView(getViewName(), model);
```

`DispatcherServlet` consults the only configured view resolver (an `InternalResourceViewResolver` object) and generates the `/WEB-INF/jsp/publicView.jsp` path.

```
<!-- View resolver -->
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass">
        <value>org.springframework.web.servlet.view.JstlView</value>
    </property>
    <property name="prefix">
        <value>/WEB-INF/jsp/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

EXAMPLE 9. `war/WEB-INF/springDispatcher-servlet.xml`

The `publicView.jsp` is a very small JSP file, it only does three things.

First, it loads all the JavaScript files needed by DWR.

Then, using the tag:`displayView` custom JSTL action and the `war/WEB-`

INF/tags/displayView.tag tag file, it renders a **static** page. The displayView.tag tag file is nothing but a special kind of JSP file. It requires an attribute called view. This is the same view object which was put in the ModelAndView Map by the controller.

```
<%@attribute name="view" ↵
    type="com.serotonin.mango.view.View" ↵
    required="true"%>
```

EXAMPLE 10. war/WEB-INF/tags/displayView.tag

displayView.tag loops over the ViewComponents found in the page to be rendered, and generates HTML code according to the type of the ViewComponent. In the example used now, the one and only ViewComponent which is in the requested anonymous view, is a virtual DataPoint, which randomly generates numbers. This DataPoint is represented by the simplest of the ViewComponents, a com.serotonin.mango.view.component.SimplePointComponent, so after calling the tag:pointComponent custom action (still from the displayView.tag),

```
<c:otherwise>
    <tag:pointComponent vc="{vc}"/>
</c:otherwise>
```

the war/WEB-INF/tags/pointComponent.tag custom tag file is rendered. This is only a static template **without** the actual values of the virtual DataPoint. Those values will arrive when the DWR framework dynamically updates this static page.

And as the last step, publicView.jsp starts a longPoll using DWR.

```
<script type="text/javascript">
    mango.view.initAnonymousView({view.id});
    dojo.addOnLoad(mango.header.longPoll.start);
</script>
```

EXAMPLE 11. war/WEB-INF/jsp/publicView.jsp

A longPoll, as its name suggests, is nothing but a function which periodically polls the server to check if there is new data. war/resources/header.js is the file which contains all the functions implementing the longPoll mechanism.

After initialization, the DWR framework calls the `MiscDwr.doLongPoll(mango.header.longPoll.pollCB)` function. `MiscDwr` is the exported JavaScript interface of the `com.serotoninmango.web.dwr.MiscDwr` Java class, so the above function call, in effect, calls into a `MiscDwr` Java object.

The `MiscDwr.doLongPoll()` Java function, among other things, is able to retrieve the contents of a `WatchList`, a `View` or the details of a `DataPoint`. This time, though, its task is to find all the values of the components of the anonymous view identified by `viewId 1`, which have changed since the last poll.

```
List<ViewComponentState> newStates;
newStates = viewDwr.getViewPointDataAnon(↪
    pollRequest.getAnonViewId());
```

After using some convoluted mechanisms to determine which `ViewComponents` have changed, it packs all the important data in a response `Map` object.

```
List<ViewComponentState> differentStates = ↪
    new ArrayList<ViewComponentState>();
// find out which newState from the newStates
// list have changed
response.put("viewStates", differentStates);
```

```
EXAMPLE 12. /src/com/serotonin/mango/web/mvc/↪
controller/PublicViewController
```

The result of the call (the response object) is given to the `mango.header.longPoll.pollCB()` JavaScript callback function.

The original Java `Map` object is converted by DWR to a JavaScript object called `response`. It only has one `viewStates` attribute, which is an array of JavaScript objects. In this example there is only one `ViewComponent`, the virtual `DataPoint`, which randomly generates numbers. The response object has an attribute called `response.viewStates[0].content` with the textual value of “`8.363517`”, and exactly this is the text which has to be injected into the – so far – static HTML page.

`pollCB()` calls the `setData()` function found in `/war/resources/view.js`,

```
if (response.viewStates)
    mango.view.setData(response.viewStates);
```

which in turn calls `mango.view.setContent()`, which finally overwrites the `innerHTML` property of the selected DOM element.

```
$("#c"+ state.id +"Content").innerHTML = ↵
    state.content;
```

If the `DataPoint` is settable or the response contains additional messages, then the `setData()` function injects more HTML code into the page, in a similar fashion described above.

Otherwise the `setContent()` and `setData()` functions end their jobs and `pollCB()` continues the polling loop by asking the server again:

```
// Poll again immediately.
mango.header.longPoll.poll();
```

EXAMPLE 13. `/war/resources/header.js`

4 DATABASE STRUCTURE IN MANGO

4.1 Tables visualized

Mango by default uses a built-in Derby database. This, however, can be easily converted into a MySQL database, as described in the Mango forum (Converting from Derby to MySQL 2009).

The `db` directory contains two scripts (`createTables-derby.sql`, `createTables-mysql.sql`) for creating the database schema. One is for Derby, the other is for MySQL. Both create the same database structure, the only difference is the syntax they use.

One goal of the project was to research the possibility of directly accessing the data

residing in the database, thereby circumventing Mango. For this, an overview of the most important tables – and the classes managing those tables now follows.

Mango uses 31 tables in its database. The relations between these tables can be easily visualized by an appropriate software, for example phpMyAdmin. PhpMyAdmin is an easy to use, open source software which can be downloaded from the <http://www.phpmyadmin.net> web site. Figure 8 shows some of the most important – regarding the goal of the project – tables and the relations in between them.

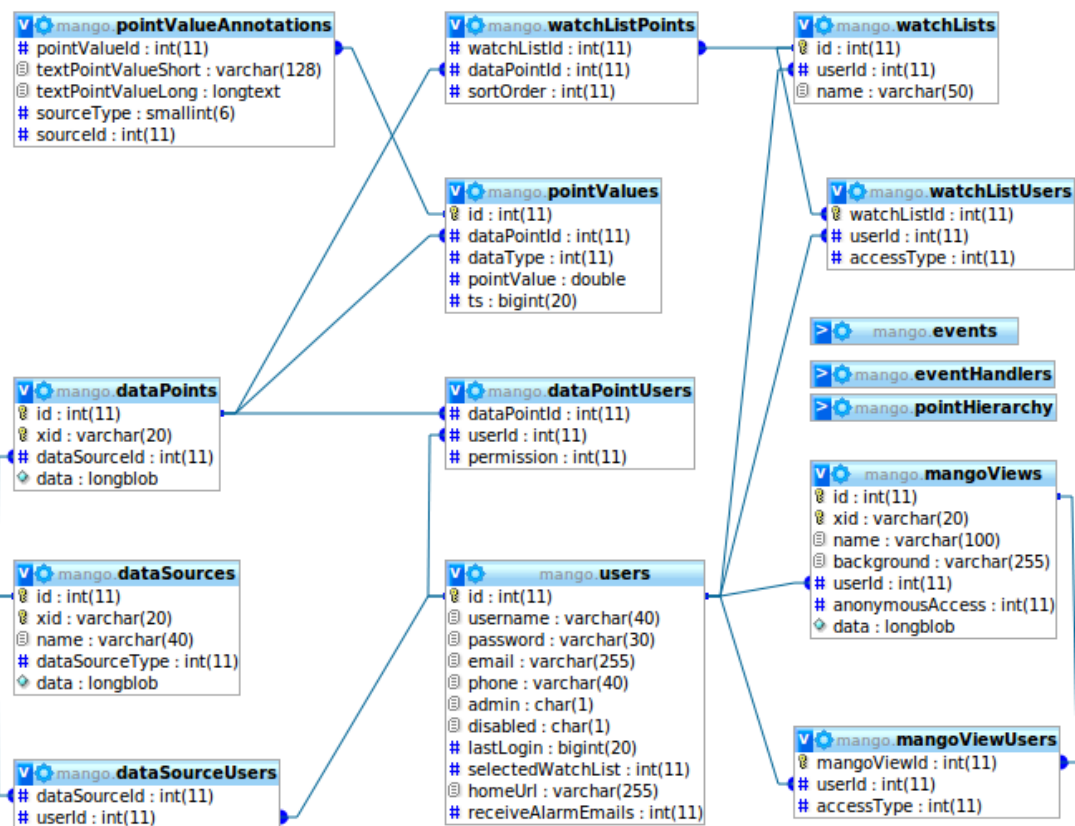


FIGURE 8. The most important tables in Mango

4.2 Data access layer

Mango – being a properly written software – uses a data access layer for abstracting away the access of data. The classes implementing the data access mechanism can be found in the `src/com/serotonin/mango/db/dao` directory. Inspecting these files, one can understand how Mango treats the collected data.

A typical DataSource is shown on Figure 9.

FIGURE 9. The details of an HTTP Retriever DataSource

A peculiarity of Mango is that in many cases Java objects are serialized into the database as-is, that is without saving the attributes of an object into specific columns of the database. An example for this is the data column of the dataSources table. Its type is longblob and its content is a `com.serotonin.mango.vo.dataSource.DataSourceV0` object:

```
<!-- View resolver -->
<bean id="viewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass">
    <value>org.springframework.web.servlet.view.JstlView</value>
  </property>
  <property name="prefix">
    <value>/WEB-INF/jsp/</value>
  </property>
  <property name="suffix">
    <value>.jsp</value>
  </property>
</bean>
```

EXAMPLE 14. `src/com/serotonin/mango/db/dao/DataSourceDao.java`

The `writeObject()` method of the `DataSourceV0` class defines how the `DataSourceV0` object gets serialized:

```
private void writeObject(ObjectOutputStream out) throws IOException {
  out.writeInt(version);
  out.writeBoolean(enabled);
  out.writeObject(alarmLevels);
}
```

EXAMPLE 15. `src/com/serotonin/mango/vo/`

dataSource/DataSourceV0.java

Similarly, some of the important attributes of a DataPointV0 object (e.g. its name) are also serialized instead of inserted separately into the columns of the dataPoints table:

```
private void insertDataPoint(final DataPointV0 dp) {
    // ...

    // Insert the main data point record.
    dp.setId(doInsert(
        "insert into dataPoints (xid, dataSourceId, data) values (?,?/?)",
        new Object[] {dp.getXid(), dp.getDataSourceId(),
            SerializationHelper.writeObject(dp)},
        new int[] {Types.VARCHAR, Types.INTEGER, Types.BLOB}));

    // ...
}
```

EXAMPLE 16. src/com/serotonin/mango/db/dao/DataPointDao.java

The serialized DataPointV0 defines its serialization in its writeObject() method:

```
private void writeObject(ObjectOutputStream out)
    throws IOException {
    out.writeInt(version);
    SerializationHelper.writeSafeUTF(out, name);
    out.writeBoolean(enabled);
    out.writeInt(pointFolderId);
    out.writeInt(loggingType);
    out.writeInt(intervalLoggingPeriodType);
    out.writeInt(intervalLoggingPeriod);
    out.writeInt(intervalLoggingType);
    out.writeDouble(tolerance);
    out.writeInt(purgeType);
    out.writeInt(purgePeriod);
    out.writeObject(textRenderer);
    out.writeObject(chartRenderer);
    out.writeObject(pointLocator);
    out.writeInt(defaultCacheSize);
}
```

EXAMPLE 17. src/com/serotonin/mango/vo/DataPointV0.java

The situation is somehow easier in the case of actual point values. Point values are represented by com.serotonin.mango.rt.dataImage.PointValueTime objects and stored in the pointValues and pointValueAnnotations tables with the help of a com.serotonin.mango.db.dao.PointValueDao object. The pointValues table only stores numerical data in its pointValue column. Images are stored in the

file system, while textual data in the `pointValueAnnotations` table.

To illustrate the above mechanism, here follows an interactive session with the built-in Derby database. For this to work, one has to download the command line tool called `ij` from the Derby web site (http://db.apache.org/derby/derby_downloads.html). This program is used to access the Derby database. Similarly, the MySQL database can be accessed by using the `mysql` command line tool. The connection string used in the `connect` command on the first line of the example can be found on the first line of the `mangoDB/service.properties` file in the deployed application (that is, under `tomcat/webapps`).

```
ij> connect 'jdbc:derby:/path/to/mango/database/mangoDB';

ij> select ds.name as DataSource_Name, ds.xid as DataSource_Xid,
dp.xid as DataPoint_Xid, pv.pointValue as PointValue
from pointValues pv, dataPoints dp, dataSources ds
where pv.dataPointId = dp.id and dp.dataSourceId = ds.id
and pv.id = 14054;

DATASOURCE_NAME      |DATASOURCE_XID|DATAPPOINT_XID|POINTVALUE
-----
my-virtual-data-source|DS_378647      |DP_989335      |24.1094772

1 row selected
ij>
```

EXAMPLE 18. Selecting a numerical point value

```
ij> select ds.name as DataSource_Name, ds.xid as DataSource_Xid,
dp.xid as DataPoint_Xid, pv.pointValue as PointValue,
pva.textPointValueShort as PointValue_Text
from pointValueAnnotations pva, pointValues pv,
dataPoints dp, dataSources ds
where pva.pointValueId = pv.id and pv.dataPointId = dp.id
and dp.dataSourceId = ds.id and pva.pointValueId = 15897;

DATASOURCE_NAME|DATASOURCE_XID|DATAPPOINT_XID|POINTVALUE|POINTVALUE_TEXT
-----
weather          |DS_954543      |DP_219249      |0.0        |Light Rain Showers

1 row selected
ij>
```

EXAMPLE 19. Selecting a textual point value

5 AN EXAMPLE MODIFICATION

Armed with all the knowledge presented above, nothing is easier than modifying a tiny bit of Mango.

5.1 The task

In Mango it is possible to click on an image provided by an HTTP Image DataSource.

The screenshot displays two panels from the Mango configuration tool. The top panel, titled "HTTP Image properties", contains the following fields:

- Name:** http image name
- Export ID (XID):** DS_401938
- Update period:** 1 minutes(s)
- Event alarm levels:** Information (for both Data retrieval failure and File save exception)

The bottom panel, titled "Point details", shows the configuration for a specific DataPoint:

- Name:** image point
- Export ID (XID):** DP_922959
- URL:** http://localhost:8080/mango17/uploads/55.
- Timeout (seconds):** 30
- Retries:** 3
- Scaling type:** none
- Read limit (KB):** 10000
- Webcam live feed code:**

```


This is the webcam live feed code pasted into
the "Webcam live feed code"-box when editing
the "Point details".
<br /> <br />
<script>
document.write("Javascript works!");
</script>
<br /> <br />
HTML <b>formatting</b> works!
<br /> <br />
Even <c:if test="true">JSTL</c:if> tags work!

```

On the left side of the "Point details" panel, there is a "Points" table with the following content:

Name	Data type	Status
image point	Image	

FIGURE 10. An HTTP Image DataSource and its DataPoint

weather - humidity	93.0	02:20:06
my-virtual-data-source - my-virtual-data-point	8.197638351363407	02:20:01
http image name - image point		02:19:31
weather - conditions	Overcast	02:20:06

Watch list: [Reload](#) [Logout](#)

©2006-2009 Serotonin Software Technologies Inc., all rights reserved

FIGURE 11. The clickable rendered image

Upon clicking the image, the user is taken to the `webcam_live_feed.htm` of the corresponding image. This `webcam_live_feed.htm` contains nothing but the code given in the “Webcam live feed code” box when editing the DataSource (see Figure 10). By modifying the page which renders this “Webcam live feed code”, one can allow the users to upload arbitrary pictures, effectively modifying the URL attribute of the DataPoint.

5.2 Implementation

The file `war/WEB-INF/snippet/imageValueContent.jsp` tells us which URL to invoke upon clicking the image:

```
<c:if test="${!empty point.pointLocator.webcamLiveFeedCode}">
  <a href="webcam_live_feed.htm?pointId=${point.id}"
    target="webcamLiveFeed">
</c:if>
```

EXAMPLE 20. `war/WEB-INF/snippet/imageValueContent.jsp`

The following happens when the request `webcam_live_feed.htm?pointId=1` arrives at Spring. The `DispatcherServlet` consults the handler mappings in the `war/WEB-INF/springDispatcher-servlet.xml` file to decide which controller is responsible for processing the request. For modifying the rendered JSP, one has to know which JSP page gets rendered. Again consulting the `springDispatcher-servlet.xml`, the answer is `/WEB-INF/jsp/webcamLiveFeed.jsp`:

```

<!--
    URL mappings to controllers
-->
<bean id="publicUrlMappings"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/webcam_live_feed.htm">webcamLiveFeedController</prop>
        </props>
    </property>
</bean>

<bean id="webcamLiveFeedController"
    class="com.serotonin.mango.web.mvc.controller.WebcamLiveFeedController">
    <property name="viewName"><value>webcamLiveFeed</value></property>
</bean>

<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass">
        <value>org.springframework.web.servlet.view.JstlView</value>
    </property>
    <property name="prefix"><value>/WEB-INF/jsp</value></property>
    <property name="suffix"><value>.jsp</value></property>
</bean>

```

EXAMPLE 21. war/WEB-INF/springDispatcher-servlet.xml

The first modification is to insert a file upload button into the newly found war/WEB-INF/jsp/webcamLiveFeed.jsp:

Use this form to upload a new picture and thus change the already existing one:


```

<form method="post" action="fileUpload.htm" enctype="multipart/form-data">
    <input type="file" name="fileUploaded" />
    <input type="hidden" name="pointId" value="${pointId}" />
    <input type="submit" value="Upload Picture" />
</form>

```

EXAMPLE 22. war/WEB-INF/jsp/webcamLiveFeed.jsp

For this source code to work, a line has to be added to the controller which renders the webcamLiveFeed.jsp.

```

Map<String, Object> model = new HashMap<String, Object>();
model.put("code", ((HttpImagePointLocatorVO)dp.getPointLocator())
    .getWebcamLiveFeedCode());
model.put("pointId", pointId);

```

EXAMPLE 23. src/com/serotonin/mango/web/mvc/controller/WebcamLiveFeedController.java

Now the rendered JSP will contain the pointId of the DataPoint, whose URL will be changed.

The form gets submitted to the URL `fileUpload.htm` (see Example 22) which is handled by the `ImageUploaderController`.

```
<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
<bean name="/fileUpload.htm"
      class="com.serotonin.mango.web.mvc.controller.ImageUploaderController"/>
```

EXAMPLE 24. `war/WEB-INF/springDispatcher-servlet.xml`

`ImageUploaderController` does nothing but accepts a `MultipartHttpServletRequest` request (this means file uploading), saves the file into a directory specified by the variable `uploadDirectory`, then changes the URL of the `DataPoint` to point at that newly uploaded image and finally it renders the `war/WEB-INF/jsp/webcamLiveFeed.jsp` whose logical view name is `webcam_live_feed.htm`.

```
public class ImageUploaderController extends AbstractController {

    private static final String uploadDirectory = "uploads";
    private int nextImageId = -1;

    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest request,
        HttpServletResponse res) throws Exception {

        if (!(request instanceof MultipartHttpServletRequest)) {
            res.setContentType("text/plain");
            res.sendError(HttpServletResponse.SC_BAD_REQUEST,
                "Expected multipart request");
            return null;
        }

        MultipartHttpServletRequest multipartRequest =
            (MultipartHttpServletRequest)request;
        MultipartFile file = multipartRequest.getFile("fileUploaded");

        String pathToUploadDirectory = request.getSession().
            getServletContext().getRealPath(uploadDirectory);
        String relativeFilepath = saveFile(file, pathToUploadDirectory);

        String pointIdStr = request.getParameter("pointId");
        if (pointIdStr != null) {
            int pointId = Integer.parseInt(pointIdStr);
            changeUrlTo(relativeFilepath, pointId, request);
        }

        String redirectUrl = "/webcam_live_feed.htm?pointId=" + pointIdStr;
        RedirectView redirectView = new RedirectView(redirectUrl, true);
        return new ModelAndView(redirectView);
    }
}
```

EXAMPLE 25. `src/com/serotonin/mango/web/mvc/controller/ImageUploaderController.java`

The interesting part is finding the DataPoint and changing its URL:

```
private void changeUrlTo(String relativeFilepath, int pointId,
    HttpServletRequest request) {
    String newUrl = "http://"
        + request.getServerName() // "localhost"
        + ":"
        + String.valueOf(request.getServerPort()) // "8080"
        + request.getContextPath() // "/mango17"
        + "/"
        + relativeFilepath; // "uploads/12.jpg"

    DataPointVO dataPoint = (new DataPointDao()).getDataPoint(pointId);
    HttpImagePointLocatorVO pointLocator =
        (HttpImagePointLocatorVO) dataPoint.getPointLocator();
    pointLocator.setUrl(newUrl);
    Common.ctx.getRuntimeManager().saveDataPoint(dataPoint);
}
```

EXAMPLE 26. src/com/serotonin/mango/web/mvc/
controller/ImageUploaderController.java

6 CONCLUSION

When the author of Mango chose to save data by directly serializing Java objects into a BLOB column, he might have thought that in this way the use of different kind of DataPoints will be easier. Though this solution might have made his life easier, it definitely does not ease the task of using the collected data from an external, possibly non-Java application.

Fortunately, Mango provides – beside the Java serialization – an opportunity to export the data into JSON format. One possible solution might be to write a controller which can export arbitrary Java object into JSON format on the fly. The writing of such a controller should not be a big challenge. The createExportData() function in com.serotonin.mango.web.dwr.ExportDwr can provide a good basis for this. Other easier solutions may exist, but finding them certainly needs more investigation.

Other than the previously mentioned obstacle, Mango has been found to be a highly configurable system. After spending the time it deserves to be fully explored, one can customize the software according to his or her very own needs.

REFERENCES

Converting from Derby to MySQL. Accessed on 11 November 2009.

<http://mango.serotoninsoftware.com/forum/posts/list/185.page>

Ladd, S., Donald, K. 2006. Expert Spring MVC and Web Flow. Apress.

Machacek, J., Vukotic A., Chakraborty A. & Ditt J. 2008. Pro Spring 2.5. Apress.

Machine To Machine, White paper. Accessed on 11 November 2009.

http://www.steria.com/documents/File/WP_Machine-to-Machine_M2M.pdf

Mango, open source M2M. Accessed on 11 November 2009.

<http://mango.serotoninsoftware.com>

Martin, B. 1996. The Open-Closed Principle. Accessed on 11 November 2009.

<http://www.objectmentor.com/resources/articles/ocp.pdf>

Open Source Initiative. Accessed on 11 November 2009.

<http://www.opensource.org>

Spring Web MVC framework. Accessed on 11 November 2009.

<http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html>

Updating Dojo. Accessed on 11 November 2009.

<http://mango.serotoninsoftware.com/forum/posts/list/143.page>

Walls, C. 2008. Spring in action. 2nd edition. Manning.