JOGGLER: DATA HARVEST AND ANALYSIS TOOL

ONDREJ NEBESKY

Department of Computer Science

JOGGLER: DATA HARVEST AND ANALYSIS TOOL

by

ONDREJ NEBESKY

PROJECT

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

May 2011

Abstract

Increase in use of stand-alone systems for recording data has made data harvesting across several fields easier and faster. However, raw data collected from such systems need to be manipulated and processed to enable meaningful analysis. Although data are readily available, one major issue concerning analysts and scientists is collation of data from various sources. Without a standard data format, scientists and analysts are required to put in resources to bring in data from multiple sources together. The problem is aggravated when a particular data source changes its data format.

This project introduces the Joggler framework for a collection of services and interfaces to enable analysts to quickly process data and bring them to other application with minimum of coding. The Joggler framework has been demonstrated on various projects, including weather data visualization and data visualization from the Jonarda experimental site.

Table of Contents

Chapter 1

Introduction

1.1 Problem

The work described here is motivated by two examples. The first example concerns the use of weather data for climate analysis. In the past few decades, several on-line services have begun providing weather data collected in United States, including Weather Underground (WU [1]) and National Oceanic and Atmospheric Administration (NOAA [2]). Each data set from those servers contain data for one day, month or year for one weather station. One issue with these data arises from the need to deal with the incompatible data formats.

The second example is a recent Cyber-ShARE [3] project for visualizing data from the Jornada [4] experimental site, which is located in New Mexico state. Data are being collected from various sensors placed on a tower and a moving tramline. This project needs to display processed sensor data on a map. A Flex-based map application connects to many web-services to load different layers, such as map images and sensor locations. There is no simple way to publish data that is updated frequently. We want to visualize our data on a production Flex-based website without having to write data processing code on the client side. Processing the data might be especially difficult when statistical functions are involved. A single web accessible data file containing already processed data works well when there is no need to update the data.

A production site should have fast response. It cannot afford to run complex data processing in the background or download the data from its original storage every time the data is requested. Daily data on the NOAA server [5] for one weather station for one year is typically 50KB (8KB compressed) in CSV (Comma Separated Values) format and 180KB in XML format. When accessing data for multiple cities for several years, downloaded files can easily reach several megabytes.

1.2 General Approach

Joggler is a collection of web-services and interfaces that provide client applications with data storage, access, and processing capabilities. Joggler supports two types of users: programmers who access the browser-based Joggler user interface in order to configure web-services used in their applications, and end users who access data provided by Joggler to the client application.

The output specification for the web-services (see section 2.4) is configured by the programmer. For example, we want to produce a graph of precipitation and spatial data created by the R software 3^{rd} party library and publish it a via web-service. The R code, database query and other scripts should be fully configurable through the web interface without modifying Joggler code. This approach allows us to create many different web services fitting exactly to consumer application needs.

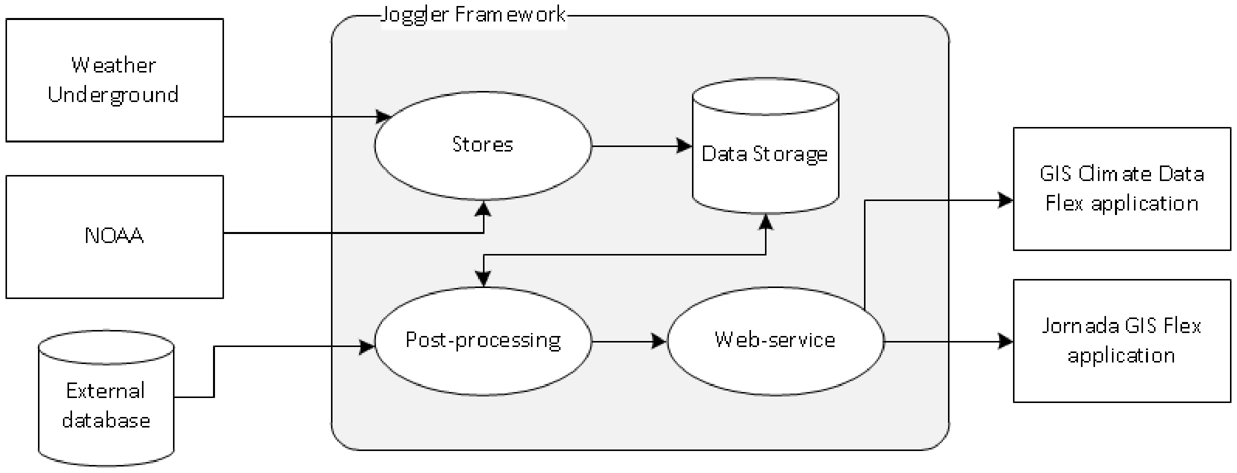


Figure 1.1: Joggler Framework with example clients

1.3 Results

Applications of this software providing proof of concept include the Jornada GIS software and a climate data Flex application. Joggler generates, for example, climate graphs using R and enables users to run SQL queries on up-to-date data. A prototype Flex application is gathering climate data through Joggler and displaying it on a map. However, the tool has even greater value for the Jornada site, which is currently being developed. Database caching is being used to upload processed data into a database, which is automatically published via a web-service to a Geographic Information System (GIS) Flex application and rendered into a graph. The tool is installed for Cyber-ShARE groups.

Chapter 2

Background

This chapter is a short introduction to technologies and software products used in Joggler. These include the Yii framework [7], the R statistical software package [11], the PostgresSQL database [12], web services [14], and Flex [13].

2.1 Yii Framework

Joggler is a PHP-based [6] application running on an Apache server using Yii Framework [7]. Yii Framework is an easily extendible, object-based framework that comes with components for both data manipulation and user interfaces. The framework is light and fast compared to well-known Zend [8] or Cake [9] frameworks. Yii Framework uses a clean MVC (Model View Controller) architecture. All the models are inherited from the built-in *ActiveRecord* class, which enables developers to load and store objects to the database without using SQL language and defining relations through database foreign keys.

Controllers take input from URL and POST requests and run actions. Each action renders one or more views.

Yii Framework extensions are stored in an extensions directory along with shared classes used in the rest of the application (*app components* in Figure 2.1). The folder includes for example a WYSIWYG editor, Javascript jQuery libraries, menu components, form components and shared classes used for debugging, authentication, and unit conversion.

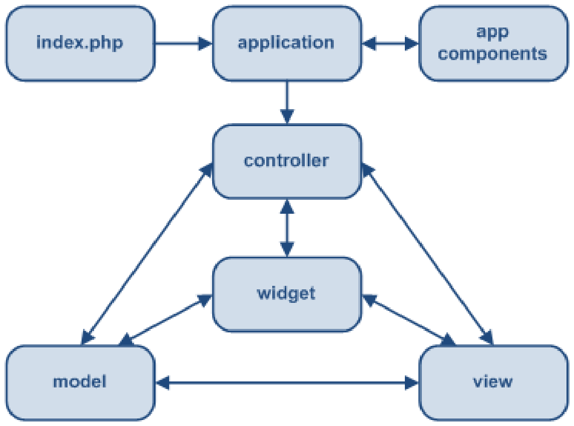


Figure 2.1: Yii Framework Architecture

2.2 R statistical software

R is powerful statistical software used for data processing and plotting. An external binary file is called when data is ready. R runs a user-specified script and may return a graph in PNG format or text output. R is extendible by many packages and can create complex graphs using minimum code.

Here is an example R code rendering a graph displayed in Figure 2.2 from Jornada NDVI tramline data:

```
png()
ndvi<-read.table("[data]", header=T, sep=",")
plot(ndvi$X_1, type="l", col="blue", xlab="Date", ylab="NDVI", ylim=c(0,0.5))
lines(ndvi$X_55, type="l", col="red")
lines(ndvi$X_110, type="l", col="green")
legend("topright", inset=.05, col=c("blue", "red", "green"),
title="Tram Position (m)", c("1", "55", "110"), fill=c("blue", "red", "green"))
```

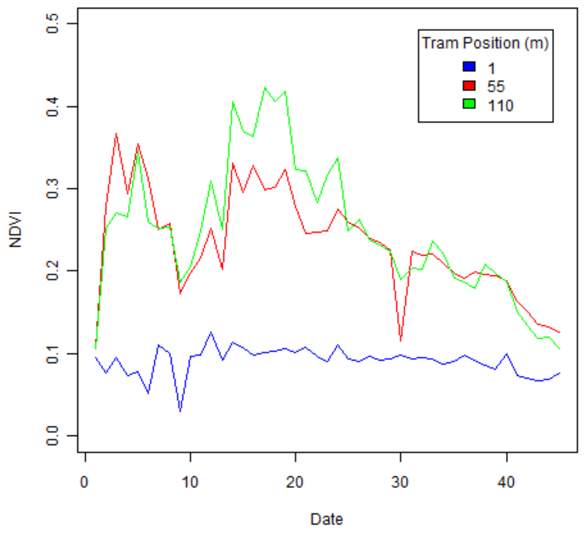


Figure 2.2: R graph example

2.3 PostgreSQL

PostgreSQL database is used to store Joggler application data structures and also recommended as a cache for data. Tables storing data from NOAA and Weather Underground can be purged, since it can be downloaded again in any moment.

2.4 Web-services

A web-service is a method of communication between two applications over a network. It uses the HTTP protocol to exchange XML messages. The Simple Object Access Protocol (SOAP) is a widely used web-service protocol, and the SOAP library is available in many programming languages. The SOAP library provides an abstract layer for programmer, so there is no need to deal with underlying XML messages. Joggler is using SOAP to get input from the client application and return processed data.

2.5 Flex

Flex an open source framework for building client side application, which communicates over network in order to load the data. Flex uses Adobe Flash technology to build rich user interface.

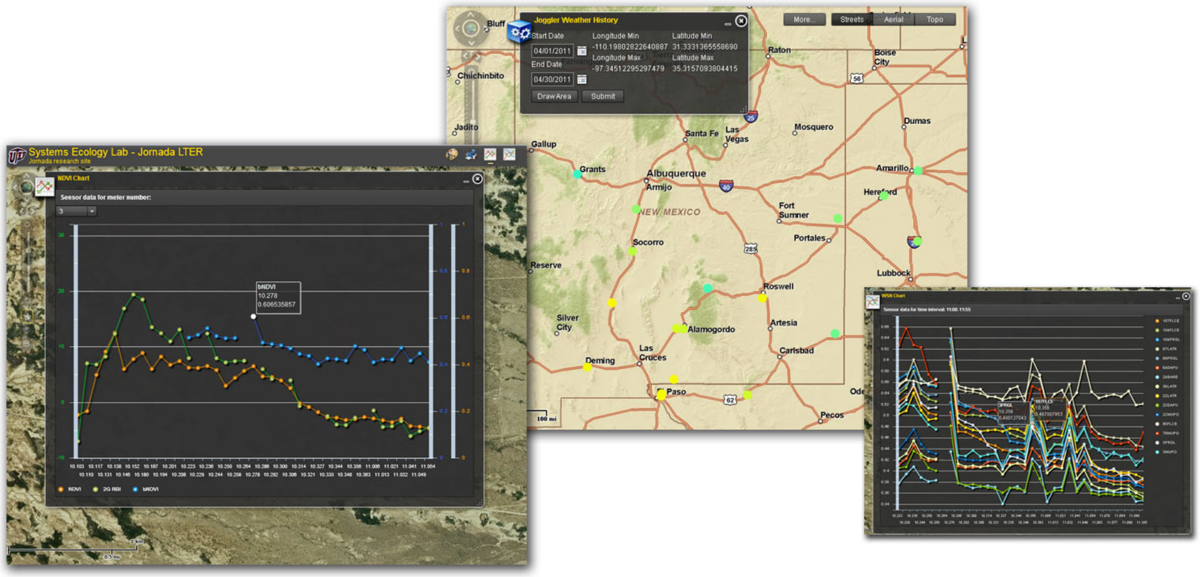


Figure 2.3: Flex user interface example

Chapter 3

Joggler

The Spring 2010 Software Engineering class at UTEP focused on tools used for climate data harvesting, gap filling, processing by statistical software and publishing via web services. Most of these projects resulted in prototypes with limited functionality, but all the teams produced detailed software requirement documents that were used as a starting point. The Joggler requirements changed during the development process, and many new requirements came with the Jornada project.

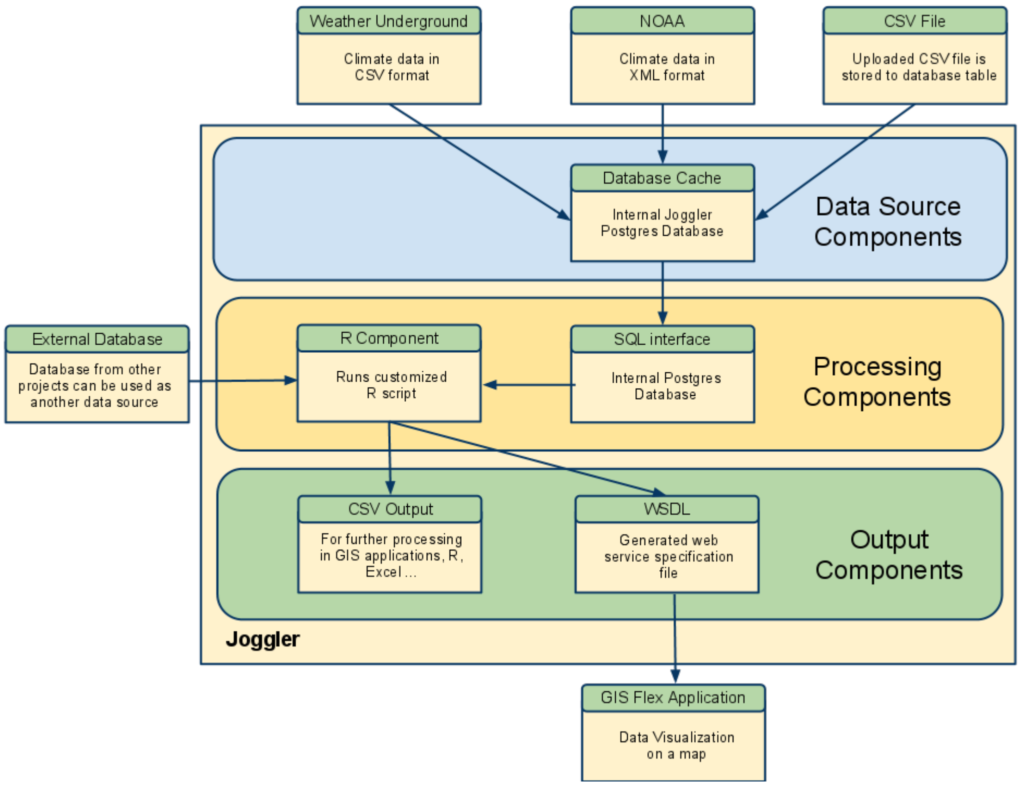
In general, the system should be able to automatically gather data, store them into database and publish via web-service using user defined SQL queries and scripts.

Main features:

- Provides easy access to near real-time climate data;
- uses a database as a cache;
- incorporates R statistical software;
- produces web services to pass the data to other applications;
- provides storage of processed data.

3.1 Architecture

Joggler uses components connected to chains to process and collect data. The data flow between Joggler components is displayed in Figure 3.1. New components are inherited from existing ones and can be easily plugged into Joggler.



Weather Underground
Climate data in CSV format
NOAA
Climate data in XML format
CSV File
Uploaded CSV file is stored to database table
Database Cache
Internal Joggler Postgres Database
Data Source Components
External Database
Database from other projects can be used as another data source
R Component
Runs customized R script
SQL interface
Internal Postgres Database
Processing Components
CSV Output
For further processing in GIS applications, R, Excel ...
WSDL
Generated web service specification file
Output Components
Joggler
GIS Flex Application
Data Visualization on a map

Figure 3.1: Component architectural diagram - the arrows indicates flow of data between Joggler components and external entities.

When a programmer creates a web-service, one or more Joggler components are used. Components pass input and data between each other and together make a chain. The chain is a final product and can be called through a web-service. Any component inside the chain can also be called from a web-service to display temporary data in the final visualization. A chain is also used as a data source for another chain. For example, there might be two chains collecting data from two servers passing data to another one, which is used to process it.

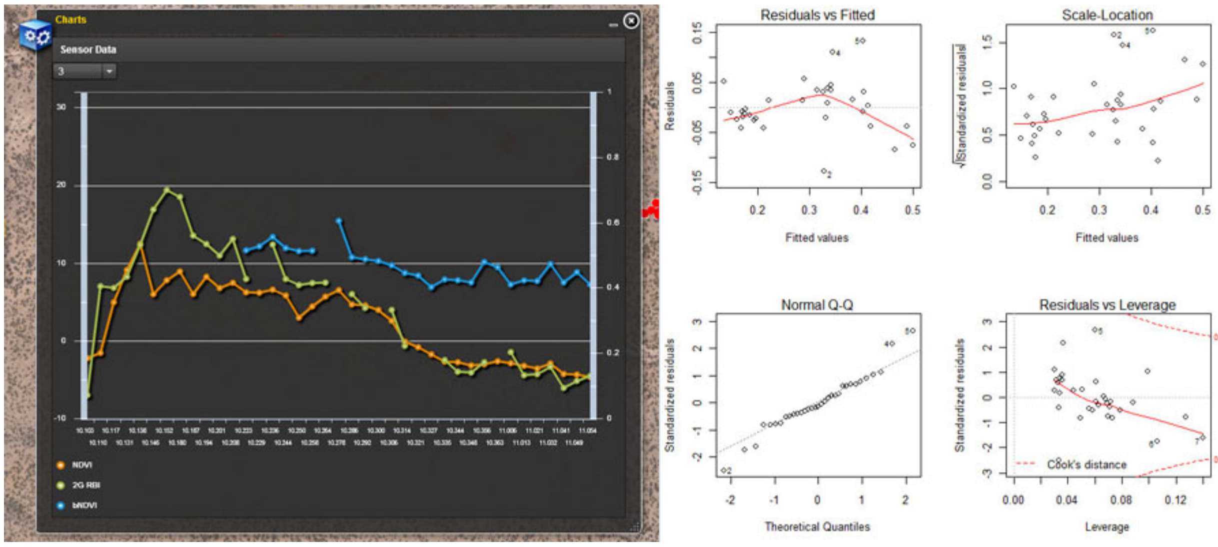


Figure 3.2: Flex vs. R graphs look and feel - The graphs produced in R (on the right) represents visualization of regression analysis of NDVI and Green Index from meter 3 from tramline at Jornada experimental site. The graph on left produced in Flex is visualizing used data.

3.2 R software integration

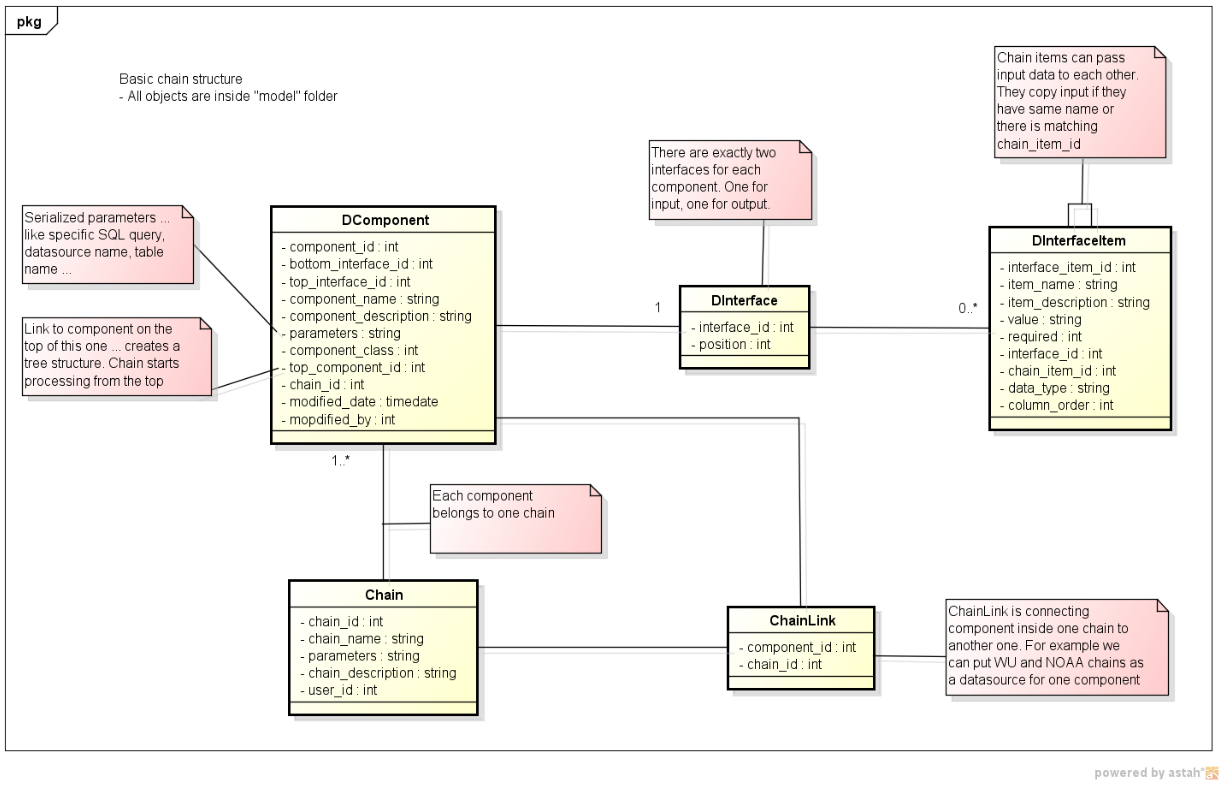
R statistical software is used inside a chain for additional data processing. Complex graphs and text output produced by R can be displayed in the client application. Programmers just need to provide R code. However, there are some limitations, which should be considered.

First, a graph produced by R is a simple image and does not allow any interactivity. For example, the user can see exact values on a graph produced in Flex by hovering a mouse over a point. On the other hand, Flex interface does not support complex graphs with trend lines. Comparison of graphs produced by Flex and R are displayed in Figure 3.2.

Second, the text output coming from R is formatted to be human readable, which makes it difficult to use in other applications. A parser is required to get values from the formatted text.

3.3 Classes and their Responsibilities

This section describes main classes used to build data collecting and processing chains. These classes implement the interface to the Model View Controller (MVC) architecture.



Basic chain structure
- All objects are inside "model" folder

Serialized parameters ... like specific SQL query, datasource name, table name ...

Link to component on the top of this one ... creates a tree structure. Chain starts processing from the top

There are exactly two interfaces for each component. One for input, one for output.

Chain items can pass input data to each other. They copy input if they have same name or there is matching chain_item_id

Each component belongs to one chain

ChainLink is connecting component inside one chain to another one. For example we can put WU and NOAA chains as a datasource for one component

Figure 3.3: Chain Structure class diagram - objects inside one data processing chain and their connections. Chain is the basic block of whole application. A component can use another chain as a data source.

DComponent

The *DComponent* is a super-class for all the Joggler components. The purpose of this class is to provide an interface to store parameters, define default values, create related objects such as interface and interface items, make connections to other components, and define user interface forms used to configure components.

DInterface + DInterfaceItem

The *DInterface* class is used to manage a collection of *DInterfaceItems*. *DInterfaceItem* is an input parameter that can be set by a web-service request or a test form. These input parameters are used as placeholders in scripts (SQL, R, PHP) or hard-coded into components. For example, *start_date* input parameter is used almost in every chain to filter data loaded from the database. Example SQL query: *SELECT * FROM table WHERE measurement_date > [start_date].*

Interfaces are linked to each other similarly as components. The reason is that input an value has to be passed to each component in the chain, and different components can use different names for the same parameter. If parameter name is the same, the interface passes the parameter automatically. However, the user can always define custom interface mapping between components.

Chain

The *Chain* class is a component container. One component belongs to one chain only. A set of components behave like one object and makes it easier to label its functionality and configure in the user interface.

ChainLink

Responsibility of this class is to connect chains together through components inside. Connected chains behavior is similar to single chain. Links allows users to reuse existing chains without copying components inside. For example, a user can attach Weather Underground data-source chain as well as NOAA data-source chain inside one component to make sure that data from both sources will be in database before a final SQL query is executed.

3.4 Database

Joggler uses multiple databases. The primary reason is that we want to use the tool on existing independent databases and data management systems. Joggler stores models to its own database. This setup also allows easier backups. Application tables have only a few kilobytes, compared to data tables, which can take much more. In addition, we can access data from various databases, such as PostgreSQL, MySQL, MSSQL and many more changing only one configuration parameter.

Separating data from the application structure also improves security. The users can run their own queries only on the data storage database without having any access to the application database. The user account can have read-only access permission guaranteeing that source data cannot be modified and in the same time have ability to create new web services inside Joggler.

However, the tool can be also used for direct data upload into database, which requires higher permissions. Tables in the data storage database are created on the fly. The user can upload Comma Separated Values (CSV) file and system consequently creates database table in this schema.

Access to the database can be a performance bottleneck. When data is not found in the database, it is downloaded from its source, inserted to database, and loaded again. Storing one record at the time might cause big delays. SQL is text oriented protocol and sends a lot of redundant data, especially when we need to store a high number of records. The solution, in this case, is a buffer, which commits 100 records in one single query.

3.5 Security

All web-services provided by Joggler are public accessible without any authentication. Authentication is used only before Joggler object is modified.

We want to make sure that parameters passed to the web-service will not expose any additional data from the server. In general, the attacker can use SQL injection vulnerability by putting SQL code inside an input parameter, which modifies original SQL query and as a result returns different data or modifies the database. Joggler is executing not only SQL queries, but also custom R and PHP code, which provides more opportunities for code injection. The attacker can potentially get control of the server.

Input values coming from the user through the client application have to be sanitized to make sure any of the used technologies cannot be attacked. The first security mechanism is *validateInput()* function inside the *DInterface* class. The function check for the data type first. Numbers and date types cannot contain any harmful characters, but string and text types need to be sanitized. We have to consider that the value might appear inside R and PHP code, from which system functions can be executed. Potentially harmful characters like brackets, variable and control characters are removed in this step. Second, special characters inside the input value are escaped by backslashes. Most of the SQL queries are executed through PDO extension, which sanitize input again.

Protection against an authenticated user is more difficult. Components execute customized R, PHP and SQL code allowing the user to access most of the files and programs on the server. This is a reason why Joggler should run on a dedicated server.

3.6 Verification

Unit tests are based on PHPUnit [10], which is standard for unit testing in PHP projects. Test driven development was used for some parts of the application. For example, database cache component keeps track of stored date intervals in separate table. These intervals cannot overlap each other, so application has to extend or merge them. There are many cases, it is not feasible to run manual testing.

The other tests are verifying basic functions, such as conversions, data type guessing, component creation process (since it is composed from several objects in database) and NOAA data source access.

Unit test unfortunately cannot run controller actions and are limited for white box testing only. They are still useful, but also functional tests should be included for more complete verification.

3.7 Deployment

Joggler runs on Apache server with PHP5 support. There is no operating system requirement. PostgreSQL database used for application should be at least version 8.1 and can be local or remote. Data storage database is separated and can be any supported database by PHP Data Object (PDO) extension. R software needs to be installed on server and should be accessible for user account running the Apache server. The reason is that R scripts might need to install additional libraries. More information about deployment is available in documentation for developers (`http://irpsrvgis47.utep.edu/documentation-developers.pdf/`). Development version is installed on `http://irpsrvgis47.utep.edu/`.

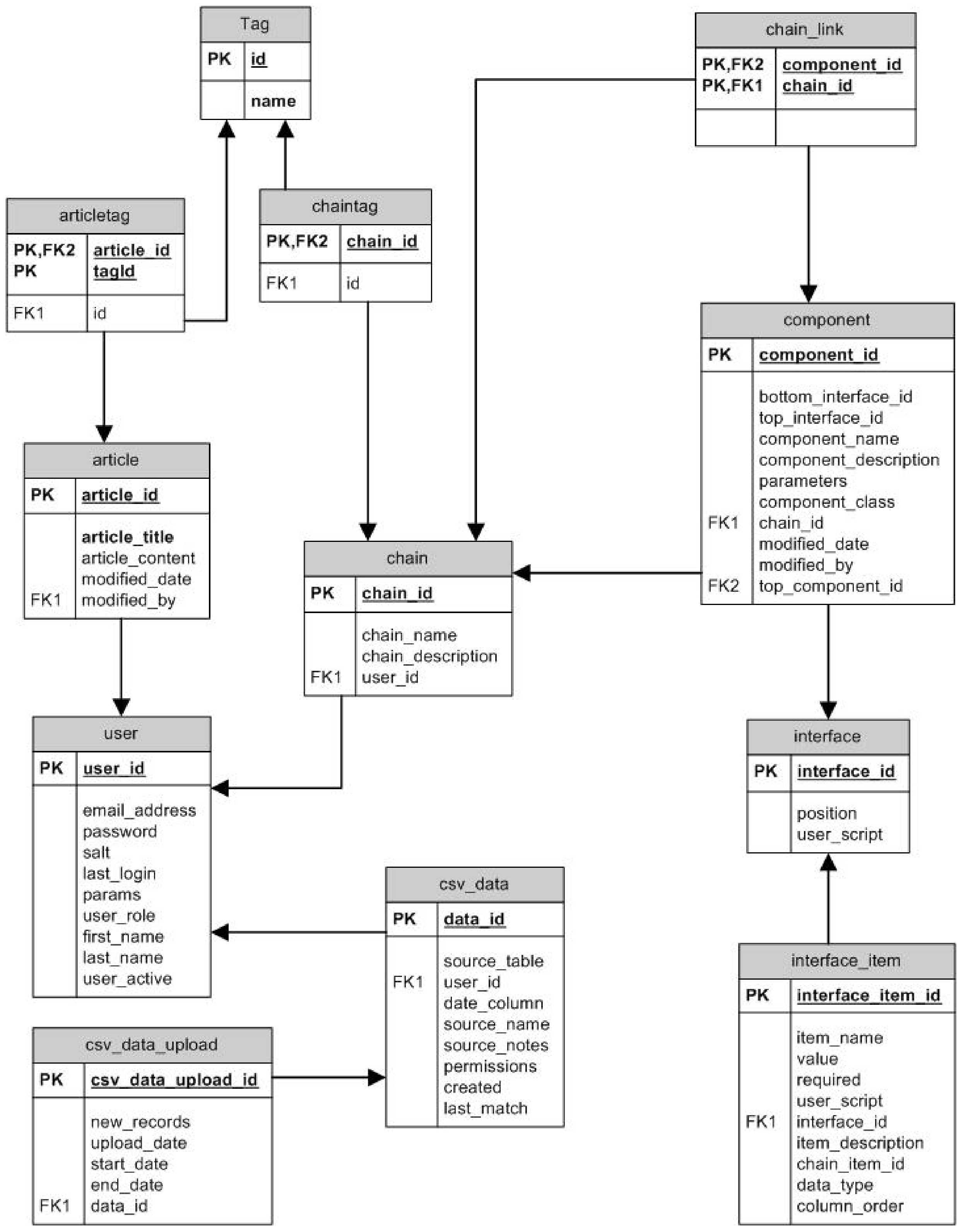
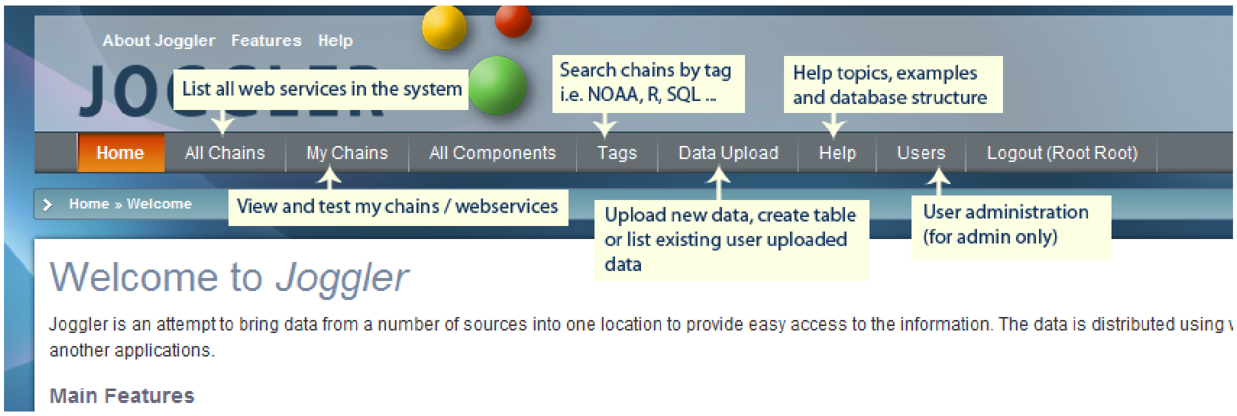


Figure 3.4: Database Schema - Application namespace

Chapter 4

The Joggler User Interface

This section walks through Joggler user interface. The section also include several screens from the application and a procedure showing how to turn excel sheet into web-service.


About Joggler
Features
Help
List all web services in the system
Search chains by tag i.e. NOAA, R, SQL ...
Help topics, examples and database structure
Home
All Chains
My Chains
All Components
Tags
Data Upload
Help
Users
Logout (Root Root)
Home » Welcome
View and test my chains / webservices
Upload new data, create table or list existing user uploaded data
User administration (for admin only)

Welcome to *Joggler*

Joggler is an attempt to bring data from a number of sources into one location to provide easy access to the information. The data is distributed using
another applications.

Main Features

Figure 4.1: User interface - main menu

4.1 Menu Structure

The following list shows complete menu structure from Figure 4.

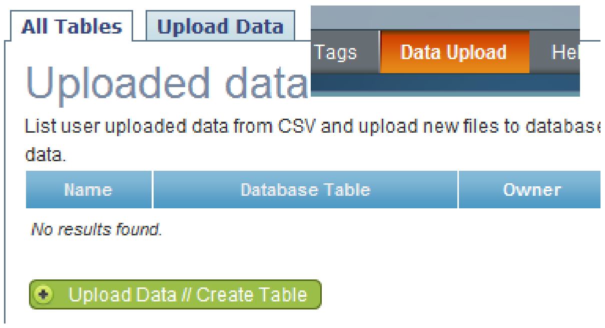
- Chain
  - *list* – view all the chains (web service accessible data)
  - *view* – view components inside the chain and connections to the other chains
  - *update* – add additional tags; tags are displayed in the main menu

- Component
  - *attributes* – manage input attributes coming into the component. Attributes are linked between each other and have data type, which is important for web services
  - *configure* – configuration of the component, which is not updated on runtime. For example, SQL and R code, as well as URL to data source is stored here.
  - *test* – run the component on provided input. Output is rendered into HTML table
- Data Upload
  - *list* – view all user uploaded tables
  - *create* – create new table based on structure of uploaded CSV file
  - *upload* – add data to exisiting table
  - *view data* – display uploaded data for one table
  - *view table* – display the table structure and all data uploads into this table
- Users
  - *create* – Create new user. Email is used as an user name, password is automatically generated
  - *profile* – View profile of current user and change password screen
- Help
  - *create article* – Add new article to help structure. Articles are stored into categories based on tags. The section contains articles related to web services, R and SQL.
  - *list* – view all articles or filter them by category. The list is also accessible from component configuration.

4.2 Data Upload

This feature allows users to upload processed data into database and publish them immediately through a web-service. First Upload of file creates new database table and consequential uploads of updated data are only adding records to this table.

The data is loaded from the database on backend when the client application request them. It can be also downloaded in CSV format from front-end.



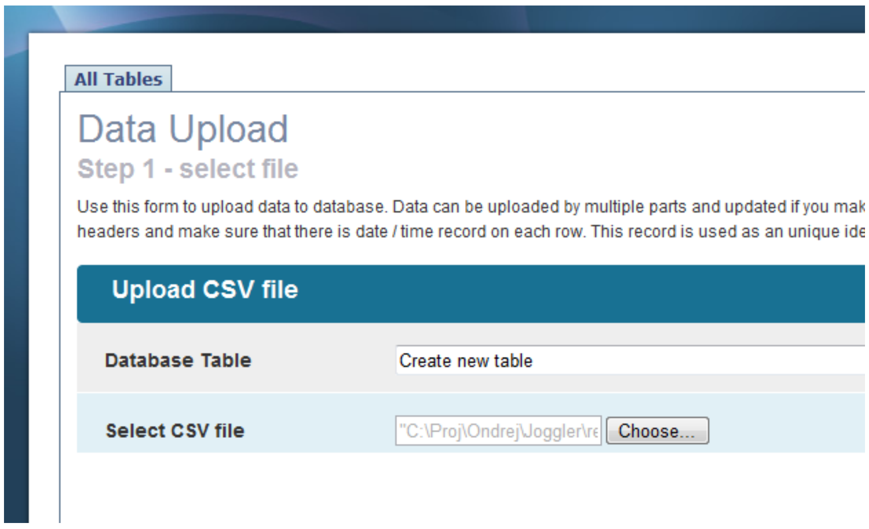
Uploaded data

List user uploaded data from CSV and upload new files to databas
data.

Name	Database Table	Owner

No results found.

Figure 4.2: View uploaded data and tables



Data Upload

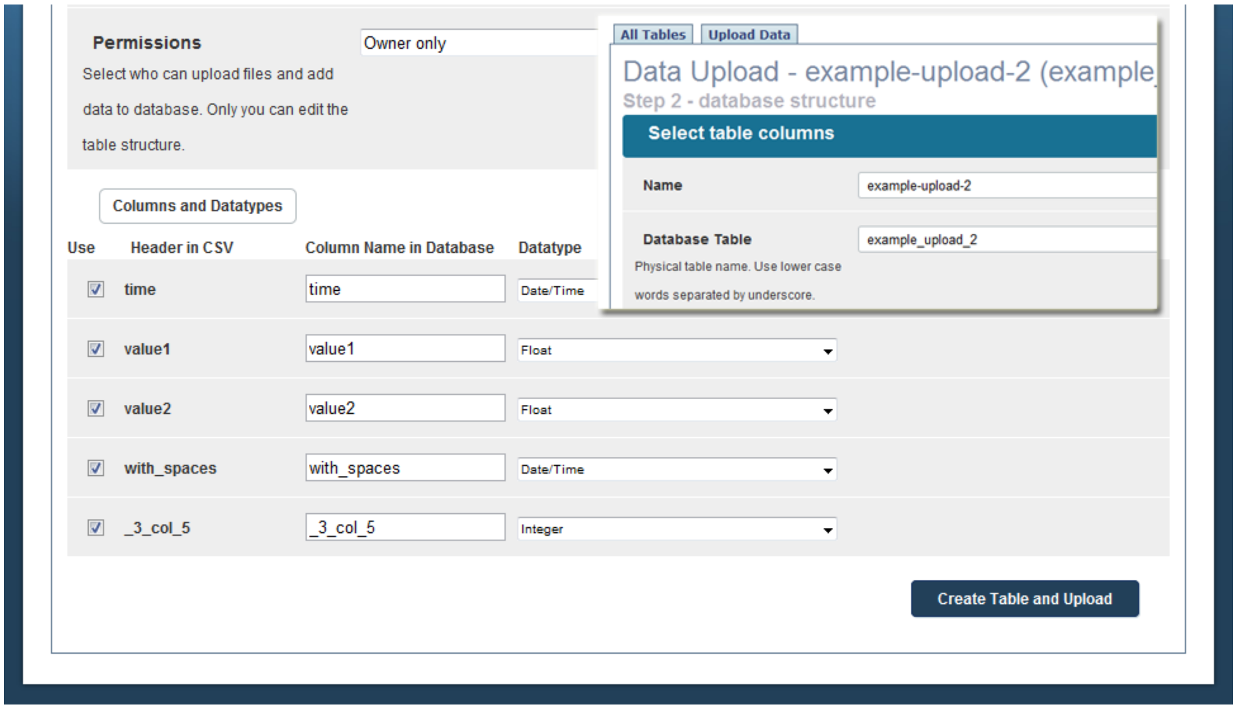
Step 1 - select file

Use this form to upload data to database. Data can be uploaded by multiple parts and updated if you mak
headers and make sure that there is date / time record on each row. This record is used as an unique ide

Upload CSV file

Database Table	Create new table
Select CSV file	"C:\Proj\Ondrej\Joggler\re

Figure 4.3: Select data file and target



All Tables | Upload Data

Data Upload - example-upload-2 (example

Step 2 - database structure

Select table columns

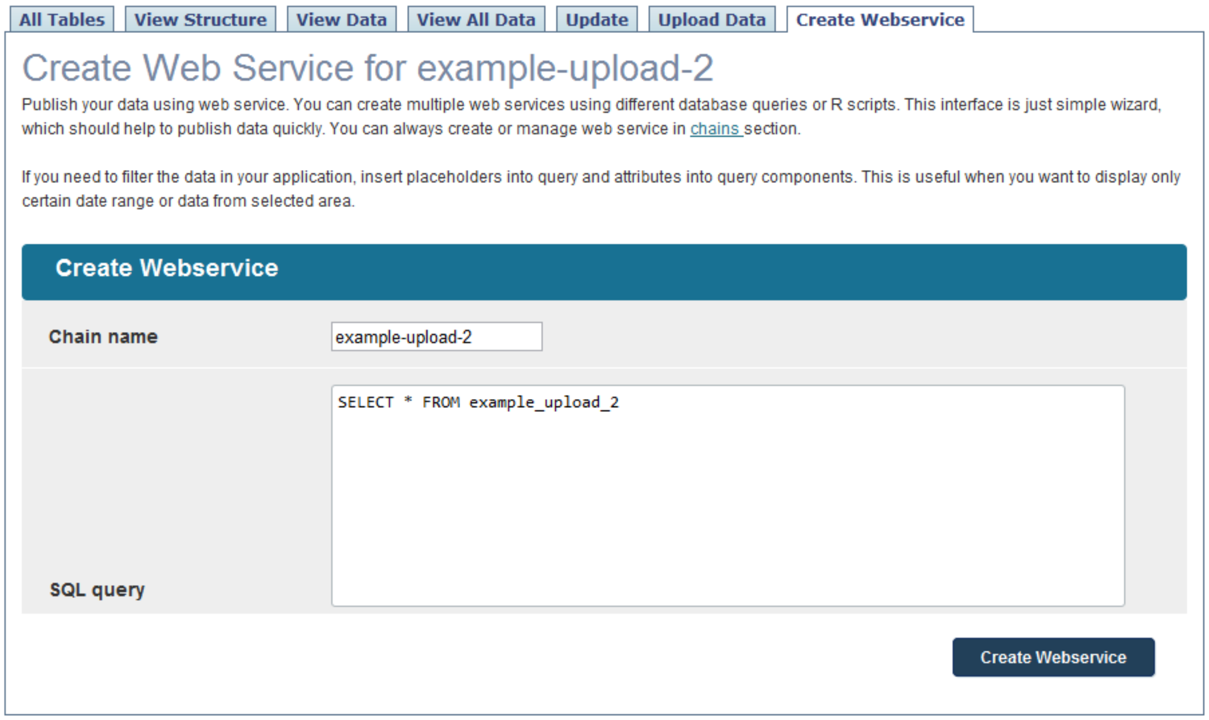
Name	example-upload-2
Database Table Physical table name. Use lower case words separated by underscore.	example_upload_2

Permissions Select who can upload files and add data to database. Only you can edit the table structure.	Owner only

Columns and Datatypes

Use	Header in CSV	Column Name in Database	Datatype
✓	time	time	Date/Time
✓	value1	value1	Float
✓	value2	value2	Float
✓	with_spaces	with_spaces	Date/Time
✓	_3_col_5	_3_col_5	Integer

Figure 4.4: Select source name, database table name and mapping between file and database. Primary key for the new table should be a measurement date from the source file. Primary key will ensure that the same data uploaded multiple times are stored only once.



Create Web Service for example-upload-2

Publish your data using web service. You can create multiple web services using different database queries or R scripts. This interface is just simple wizard, which should help to publish data quickly. You can always create or manage web service in chains section.

If you need to filter the data in your application, insert placeholders into query and attributes into query components. This is useful when you want to display only certain date range or data from selected area.

Create Webservice

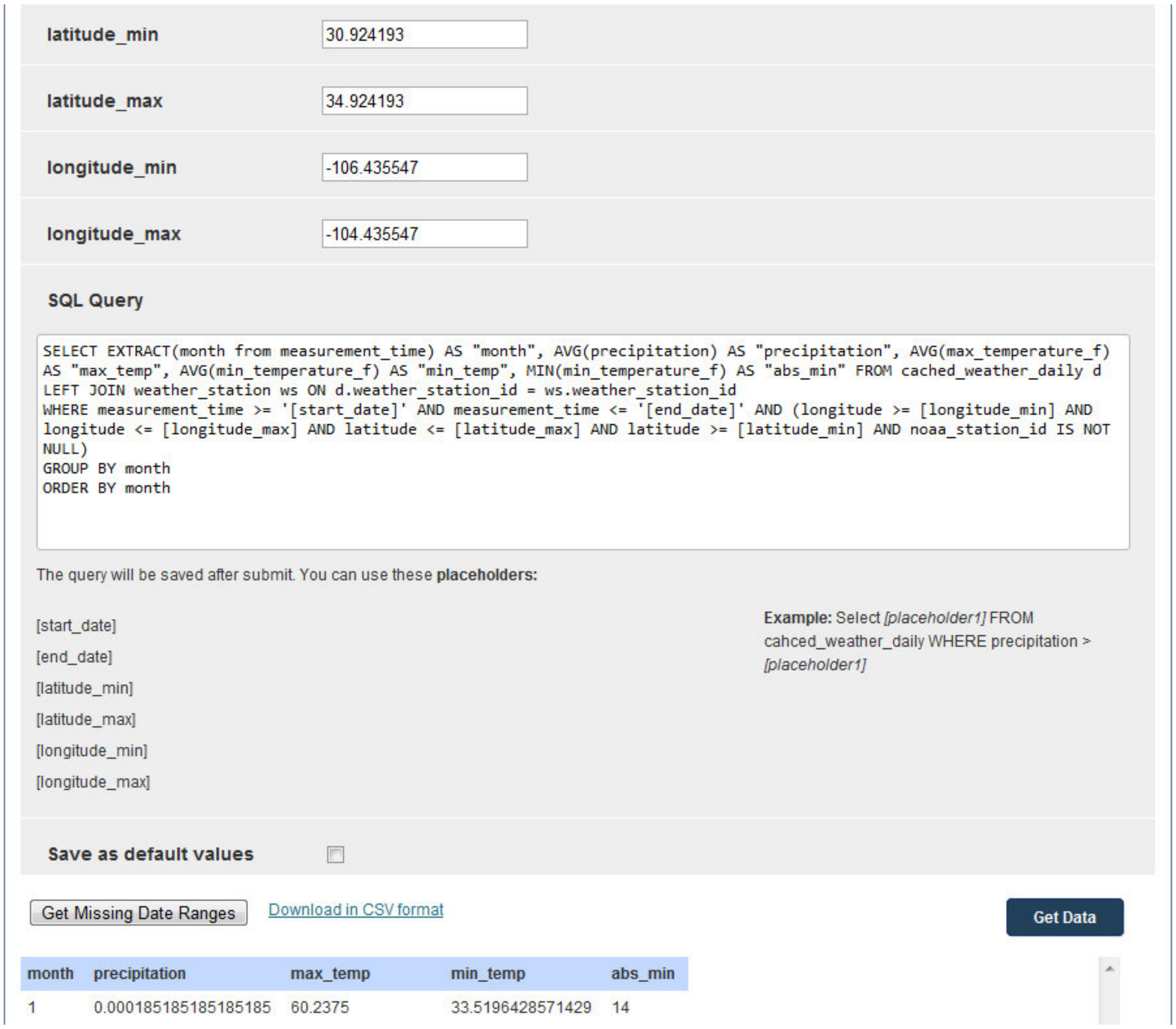
Chain name: example-upload-2

SQL query:

```
SELECT * FROM example_upload_2
```

Figure 4.5: New chain will have database query component inside. The default query will load all the data from the table, so we should add WHERE clause and specify input parameters in the next step. Web-service will return the same output as this query.

4.3 Other Screens



latitude_min	30.924193
latitude_max	34.924193
longitude_min	-106.435547
longitude_max	-104.435547

SQL Query

```
SELECT EXTRACT(month from measurement_time) AS "month", AVG(precipitation) AS "precipitation", AVG(max_temperature_f)
AS "max_temp", AVG(min_temperature_f) AS "min_temp", MIN(min_temperature_f) AS "abs_min" FROM cached_weather_daily d
LEFT JOIN weather_station ws ON d.weather_station_id = ws.weather_station_id
WHERE measurement_time >= '[start_date]' AND measurement_time <= '[end_date]' AND (longitude >= [longitude_min] AND
longitude <= [longitude_max] AND latitude <= [latitude_max] AND latitude >= [latitude_min] AND noaa_station_id IS NOT
NULL)
GROUP BY month
ORDER BY month
```

The query will be saved after submit. You can use these **placeholders:**

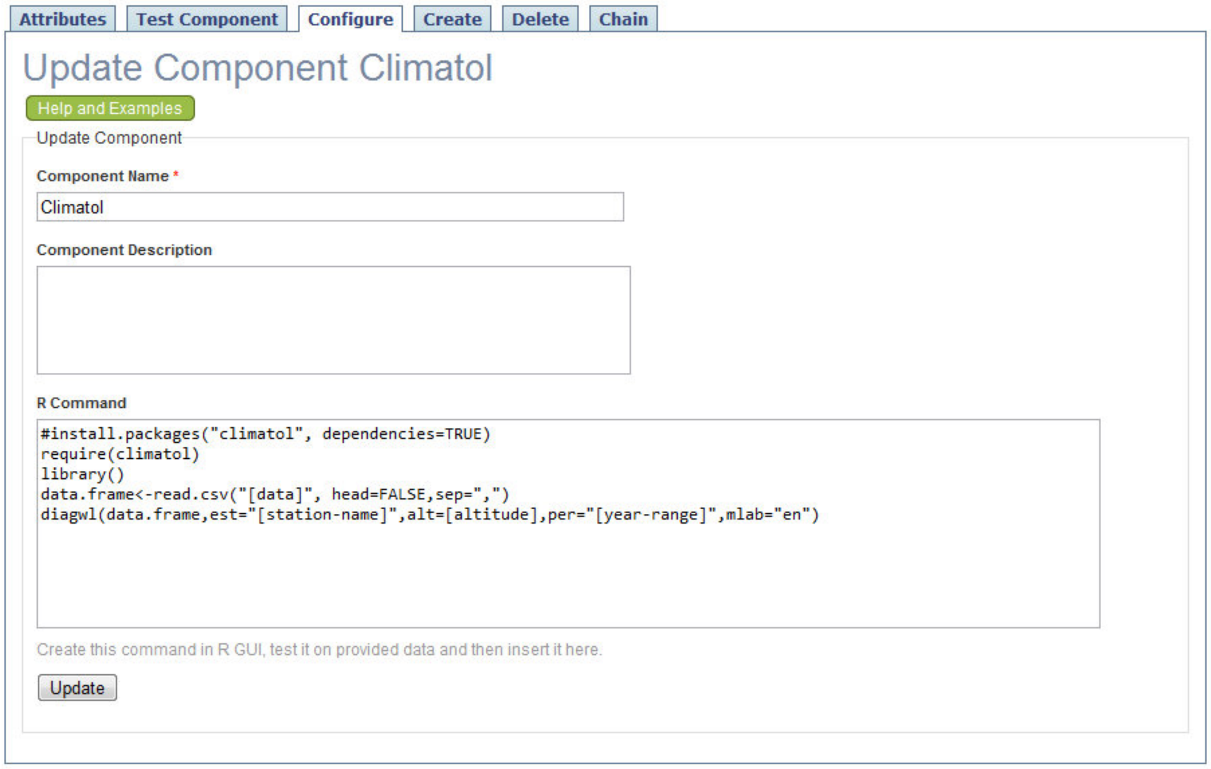
[start_date]
[end_date]
[latitude_min]
[latitude_max]
[longitude_min]
[longitude_max]

Example: Select *[placeholder1]* FROM cahced_weather_daily WHERE precipitation > *[placeholder1]*

Save as default values ☐

month	precipitation	max_temp	min_temp	abs_min
1	0.000185185185185185	60.2375	33.5196428571429	14

Figure 4.6: Test interface for database query component. SQL query is using placeholders, which are replaced by input value when the query is executed.



Update Component Climatol

Update Component

Component Name *

Climatol

Component Description

R Command

```
#install.packages("climatol", dependencies=TRUE)
require(climatol)
library()
data.frame<-read.csv("[data]", head=FALSE,sep=",")
diagwl(data.frame,est="[station-name]",alt=[altitude],per="[year-range]",mlab="en")
```

Create this command in R GUI, test it on provided data and then insert it here.

Figure 4.7: The R code is stored in component configuration. Input values, as well as data produced by other components in a chain, are available inside the code.

Chapter 5

Conclusions

The main contribution of Joggler is its capability to make data from variety of sources available thorough web-services. Data might be in the form of flat files, Excel spreadsheets or various databases, and it can be made available to visualisation applications in a matter of minutes.

With Joggler, the application developer configures data processing components to create customized output for web services. Joggler offers components with an extendible data collecting interface for retrieving data from various sources which may be cached in a database or pushed directly into web services.

Joggler has been demonstrated on two projects at UTEP. The most important is the visualization of data collected at the Jornada experimental site. At the Jornada, there are many sensors placed on a tower and a tramline. The data from all the sensors will be eventually stored into centralized database and visualized in GIS Flex-based application. The data will be rendered into various graphs, which might need to be rendered by R software.

The second Joggler application is for the collection of climate data for the Systems Ecology Laboratory at UTEP. Joggler is able to collect and process climate data from two servers and offers an interface to easily incorporate another data sources. Joggler was used to incorporated weather data into a Flex mapping application providing historical temperature data in the USA. The Flex application displays weather stations and temperatures on a map for a given day. This prototype shows that Joggler can be easily connected with the client applications. Incorporating R in a web-service processing chain opens new doors for using spatial data visualization on a map. Historical and current weather will be useful for students and scientists working on climate research.

5.1 Future Work

The tool might be very useful for automatic data harvesting from various sensors at the Jornada experimental site. The project is still in development and has a simple prototype front end available. Central data management is not ready yet.

Some of the datasets coming from the Jornada site are created by Matlab software, which can be also included in processing chain. This Matlab component, in combination with live data collecting from a server, would allow full automation of data processing and near-real-time visualization in Flex based GIS application.

Another useful extension would be support for an OPEnDAP server to create easy integration with many scientific tools. The OPEnDAP protocol makes scientific data accessible in many applications and visualization packages. A PHP OPEnDAP extension already exists, so we just need to implement it.

Grid interpolation of geo-spatial data might be useful to produce temperature and anomaly maps from the collected climate data. There are many R visualization packages operating on gridded data.

References

[1] "Weather Underground." Internet: http://www.wunderground.com/, [May 5, 2011]

[2] "National Oceanic and Atmospheric Administration."
Internet: http://www.noaa.gov/, [May 5, 2011]

[3] "Cyber-ShARE." Internet: http://cybershare.utep.edu/, [May 1, 2011]

[4] "Jornada Research Site."
Internet: http://en.wikipedia.org/wiki/Jornada_Basin_LTER, [May 3, 2011]

[5] "NOAA historical weather data archive."
Internet: ftp://ftp.ncdc.noaa.gov/pub/data/gsod/, [May 1, 2011]

[6] "PHP." Internet: http://php.net, [May 5, 2011]

[7] "Yii Framework." Internet: http://www.yiiframework.com/, [May 5, 2011]

[8] "Zend Framework." Internet: http://framework.zend.com/, [May 5, 2011]

[9] "CakePHP." Internet: http://cakephp.org/, [May 5, 2011]

[10] "PHPUnit." Internet:
https://github.com/sebastianbergmann/phpunit/, [May 5, 2011]

[11] "The R Project for Statistical Computing." Internet: http://www.r-project.org/, [May 5, 2011]

[12] "PostgreSQL: The world's most advanced open source database." Internet:
http://www.postgresql.org/, [May 5, 2011]

[13] "Flex: Open source framework, web application software development." Internet:
http://www.adobe.com/products/flex/, [May 4, 2011]

[14] "SOAP Tutorial." Internet:
`http://www.w3schools.com/soap/default.asp`, [May 5, 2011]

[15] José A. Guijarro. "Package climatol" *R packages*. [On-line].
Internet: `http://cran.r-project.org/web/packages/climatol/climatol.pdf`,
Jan. 27 2011 [Mar. 20, 2011]