

*rasdaman* as a  
Unified Storage Layer  
for THREDDS and ncWMS

by

**Jana Kohlhase**

Bachelor Thesis in Computer Science

Prof. Peter Baumann

---

Name and title of the supervisor

Date of Submission: May 11, 2014

---

Jacobs University — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicates resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.



Signature

Bremen, May 11, 2014

Place, Date

## Abstract

Many research and engineering fields have to process huge amounts of data to get meaningful results. For instance, medical imaging requires very high resolution images, geographic information systems require maps with a significant amount of data attached to each coordinate, and climate research requires datasets including huge weather data and atmospheric measurements. It is no longer a problem to procure that data, today it is what to do with the data once it is acquired. This includes the technical problems of storage and visualization and later data analytics in general. This thesis focuses on the former.

The *rasdaman* system, a pioneer array database management system, and the THREDDS data server, a dataset cataloging and metadata management system, solve complementary aspects of the storage problem. The ncWMS system, an implementation and extension of the Web Map Service standard, that includes support for most environmental datatypes, tackles visualization. But there is no overall integration of all three functionalities, all of them work with their own idiosyncratic data model and access features. In this thesis, I have remedied this by extending the THREDDS and ncWMS services to enable access to data through the *rasdaman* service, i.e., I provide an integrated approach to handle big geo-referenced data. In particular, the resulting system will combine efficient raster data storage and querying from *rasdaman* with metadata handling from THREDDS and flexible visualization from ncWMS.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Running Example: “Derek Data” . . . . .	1
1.2	Research Statement and Goals . . . . .	3
<b>2</b>	<b>Preliminaries</b>	<b>3</b>
2.1	Web Map Service (WMS) Standard . . . . .	4
2.2	Web Coverage Service (WCS) Standard . . . . .	4
2.3	ncWMS: An Implementation of WMS . . . . .	4
2.4	<i>rasdaman</i> : An Array Database Management System . . . . .	6
2.5	THREDDS: Metadata Management . . . . .	7
<b>3</b>	<b>Implementation</b>	<b>7</b>
3.1	Extending ncWMS . . . . .	8
3.2	Extending THREDDS . . . . .	10
3.3	Test Datasets . . . . .	13
<b>4</b>	<b>Related Work</b>	<b>13</b>
<b>5</b>	<b>Evaluation</b>	<b>14</b>
5.1	<i>rasdaman</i> and ncWMS . . . . .	15
5.2	<i>rasdaman</i> and THREDDS . . . . .	16
5.3	Discussion . . . . .	21
<b>6</b>	<b>Conclusion</b>	<b>21</b>

# 1 Introduction

In many research and engineering fields, research has progressed to a level where huge amounts of data are needed to make contributions to the state of the art. This phenomenon is known as “Big Data”, which also denotes an emerging high profile research field. Generating the data in the amounts needed has now become possible because of advances in technology, e.g., the sensor sector. However, once the data is generated, it needs to be stored, visualized and analyzed. We focus on the two former aspects. I will illustrate these issues with a running example now.

## 1.1 Running Example: “Derek Data”

Consider a climate scientist – Derek Data – aboard a research ship. He is responsible for gathering and analyzing data like water temperatures or changes in ocean currents and weather patterns. He gathers multiple terabytes of data daily. Actually, Derek gathers two types of data: metadata and actual data. **Metadata** is “data about data”, it describes the properties that data has. For example, the water temperature measurements Derek gathers were taken at a certain place, at a certain time, by a certain person (Derek) using sensors of particular characteristics. All this metadata is collected along with the actual measurements.

Derek has enough local disk space so that he can store the gathered data in raw form. However, in order to be able to analyze the data later on, he faces a couple of issues:

- He needs to be able to *acquire other data* from his research institute on the mainland to compare the measured data with the expected data, for example.
- He wants to *interpret the (mass of) data*. As a picture says more than a thousand words, he would ideally like to scrutinize an image enhancing the relevant data, e.g., where the temperature is overlaid over a map.
- To handle the huge amount of incoming data, he wants to *organize/categorize* it. Then, for instance, he can pick and choose whether he wants to browse all available current data or only data from days where the current was especially strong.
- Finally, he also needs to be able to *focus on specific data sections* that interest him. For example, the research ship receives weather images generated by satellite. But the ship is currently anchored off the coast of Australia, so Derek is only interested in the part of the data that depicts Australia.

Today, getting data from the mainland to Derek’s laptop aboard the research ship is no great problem. The data can simply be transmitted over the internet. However, this only allows for the transport of entire files. So if Derek needed the weather patterns over Australia for the last month, he would have to download all the files (one per day), where each covers weather patterns over the entire earth. This is a huge amount of data and transmitting it as raw byte data, the download may take days or even weeks. Instead Derek uses a clever database called *rasdaman* [RDM], which first saves the data as **raster data** (multidimensional array data, see Fig. 1 for an example) in a way that querying it is very efficient.

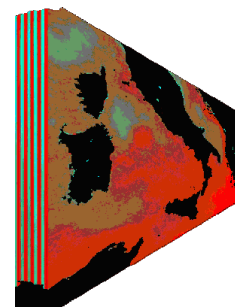


Figure 1: Layered Geographical Data (image from [GSD])

Then it allows users to query that data over the web so that only relevant data is actually transmitted.

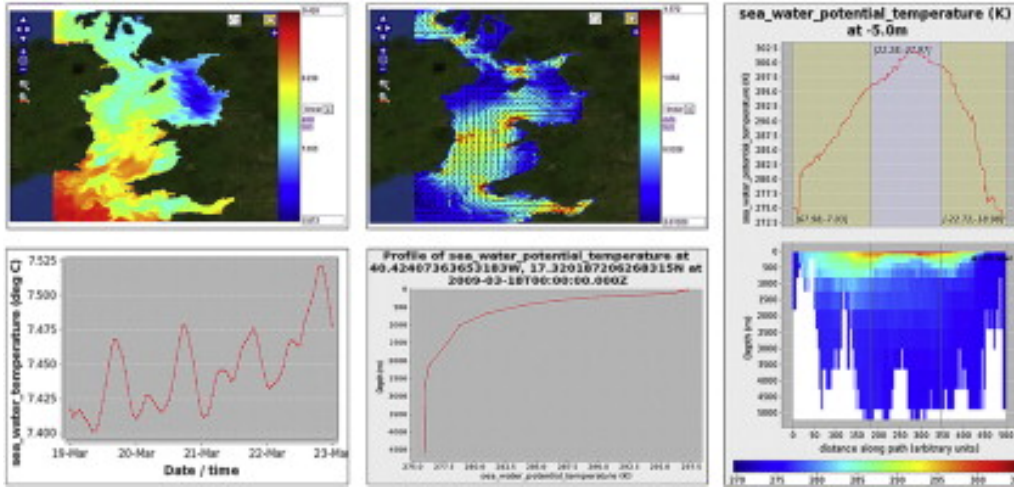


Figure 2: Examples of ncWMS Visualizations (image from [BGG<sup>+</sup>13])

*rasdaman* does not support direct visualization of the data, however. For this, Derek can use a service called ncWMS. This service allows him to visualize both local and remote geospatial data. Fig. 2 shows the result of various ncWMS visualizations. In the upper left corner, for example, we see a heat map overlaying a land map. Derek can directly grasp where exactly the hot and cold temperatures arise.

To organize his data, Derek uses a data cataloging system called THREDDS [DCD<sup>+</sup>03, Car02, THRa]. While this service allows rudimentary querying of data, it is actually primarily a metadata management service. THREDDS allows querying the metadata of several datasets at once, so that Derek can find all the datasets created by his friend (he knows they have the highest quality). Derek prefers to use THREDDS over *rasdaman* for data organisation, because data in THREDDS is more conceptually organized. THREDDS is limited to local data though, so he cannot organise the data from the mainland without first acquiring a local copy via a different service. Recently, an ncWMS component has been added to THREDDS, so Derek can now actually generate map images directly from the THREDDS catalog. Unfortunately, this component was specifically adapted for THREDDS use only and thus, Derek cannot access new features of ncWMS.

*rasdaman* and THREDDS directly allow Derek to define queries of what part of the data should be returned, *rasdaman* with many more customizable options. ncWMS has an interactive interface that allows viewing of different parts of the data. So, in principle, he can access only specific parts of the available data within all three services.

We have seen that Derek has to use a different system to address each issue, except for the last. As a consequence, he is forced to continually switch back and forth between systems. He can use THREDDS to organise his local data, but not remote. He can use ncWMS to visualize data, but he first needs to find the data locations, from THREDDS for example. Because of the recent integration, he can now organise and visualize local data seamlessly, but he misses out on new features of ncWMS. To retrieve remote data and query for parts efficiently and flexibly, he can use *rasdaman*. But he cannot organise all data, both remote and local, query it flexibly and then visualize the parts, without having to shuffle data between all 3 applications.

## 1.2 Research Statement and Goals

From a theoretical standpoint, Derek’s workflow implies a set of requirements, each of which is addressed best by a specific service:

- **Efficient querying with *rasdaman***, an array DataBase Management System (DBMS) [Bau99], that adds array support to standard relational databases, enabling users to store and query massive multidimensional data (MMD) quickly and efficiently. To allow universal accessibility and to support integration, *rasdaman* implements the relevant standards of the Open Geospatial Consortium (OGC) [OGC].
- **Organization with THREDDS**, a web server that provides metadata and data access for scientific datasets.
- **Visualization with ncWMS**, an implementation of the OGC’s Web Map Service (WMS; see section 2.1) protocol, one of the most accepted ways of visualizing geospatial data. WMS standardizes the serving of geo-referenced map images (generated by a map server) via the internet.

Each of these services solves an issue described in section 1.1. This leads me to two research questions I want to find answers for:

**RQ1** How can *rasdaman* be integrated with ncWMS such that we can use ncWMS to visualize data that was stored in *rasdaman*?

**RQ2** And how can we add *rasdaman* support to THREDDS, such that datasets stored in *rasdaman* can be inventoried in a THREDDS catalog?

The obvious solution for joining these three services and thus eliminating the need for the user to port data back and forth, would be to create one joint application of *rasdaman*, THREDDS and ncWMS. However, this is neither feasible practically nor elegant theoretically. A single application would lose some of the flexibility these 3 separate services provide and become too unwieldy to really handle with ease. Furthermore such a tight integration would lock the components to a specific version and the user out from new features. So instead, I propose to hook up *rasdaman* with THREDDS and ncWMS respectively. This will allow all 3 services to keep their flexibility, while allowing both THREDDS and ncWMS to read data directly from *rasdaman*, ultimately leading to end-to-end services.

My research contribution consists of the integration of THREDDS and ncWMS with *rasdaman*. To show that I have achieved such an integration, I will provide a demo of the end-to-end services in section 5 and provide a cognitive walkthrough based on the running example.

## 2 Preliminaries

In this section I will give an in-depth introduction to the three systems I want to integrate and the standards upon which the integration is based.

## 2.1 Web Map Service (WMS) Standard

Established in 2000 by the OGC, the **Web Map Service** [WMSa] standard specifies a simple HTTP interface for requesting geo-spatial map images from one or more databases. A typical setup consists of a server that receives requests, processes the payload and sends it back to the client that requested it. A WMS request defines the specific area that the client is interested in. There are multiple request types, each of which returns a different output format. The three main request types are:

1. **GetCapabilities** requests an XML document containing metadata on service capabilities supported by the particular server,
2. **GetMap** requests a map image for user-defined map parameters,
3. **GetFeatureInfo** requests more information on a specific pixel in a map image.

The WMS is used in geographic information systems with over 35 implementations currently in use [wmsb].

## 2.2 Web Coverage Service (WCS) Standard

The **Web Coverage Service** [WCSa, WCSb] standard, first formalized in 2003 by the OGC, specifies an interface for access to multidimensional coverage (that is geospatial information representing space/time-varying phenomena as defined in [WCSb]) data over the internet. To this end, the WCS interface specifies the following operations that may be invoked by a WCS client and performed by a WCS server:

1. **GetCapabilities** allows a client to request information about the servers capabilities and coverages offered
2. **DescribeCoverage** allows a client to request detailed metadata on selected coverages offered by the server
3. **GetCoverage** allows a client to request a coverage comprised of selected range properties. The clients return payload can consist of multiple different formats e.g. XML or TIFF.

The WCS is also used in geographic information systems and currently has about a dozen implementations [wcsc].

## 2.3 ncWMS: An Implementation of WMS

Developed by the Reading e-Science Centre at the University of Reading, UK, **ncWMS** [BGG<sup>+</sup>13, NCW], an implementation of the WMS standard, is used to visualize data without the need to download huge files or interpret complex metadata. It converts data from a number of common scientific formats into map images in many different coordinate reference systems. Most notably the **NetCDF** [CDFa] format, a self-describing, scalable, portable, appendable, and shareable format for the creation, access, and sharing of array-oriented scientific data, and CF [CFM] metadata for Climate and Forecast data is used. Moreover, ncWMS lets a user request a *customizable*, geo-referenced image



of a dataset from a server together with associated metadata. ncWMS introduces several extensions/specializations of the WMS specification to accommodate some common scientific requirements:

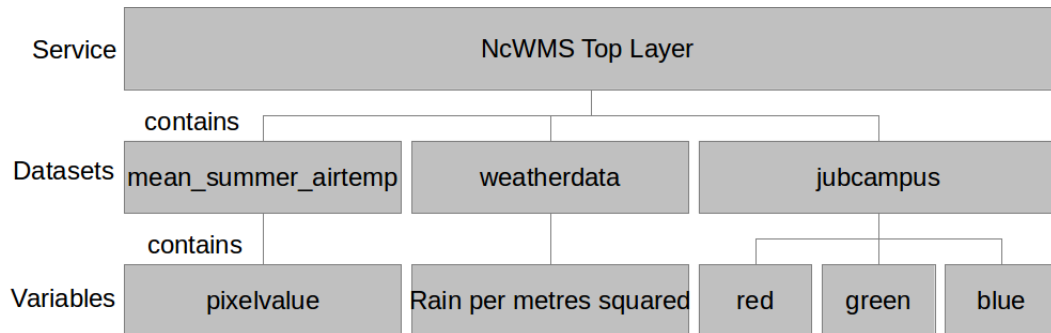


Figure 3: Example of ncWMS Layer Nesting

1. **Layer Nesting (specialization)** ncWMS is organized internally into three hierarchical layers (see Fig. 3). The top level layer represents the whole service (containing metadata on the available datasets). This layer can be “expanded” to reveal the second layer representing the datasets offered by that server. The service in Fig. 3 contains three datasets: `mean_summer_airtemp`, `weatherdata`, and `jubcampus`. Each of these datasets can again be expanded to its variables. A variable is a set of distinct data within a dataset. The `jubcampus` dataset, for example, contains the three variables `red`, `green`, and `blue`, each representing a different color band of the satellite imagery. This layer is the only layer that is actively displayed. Fig. 15 shows layer nesting in action: the navigation tree on the left directly corresponds to the layers, the image on the right shows the variable `pixelvalue` from the `mean_summer_airtemp` dataset.
2. **GetFeatureInfo (specialization)** ncWMS uses this operation not only to give the user access to the data value represented by a given map pixel, but also to allow the generation of time series plots at a given point.
3. **Custom Styling (extension)** A standard WMS implementation only gives a finite number of “styles” in which images can be generated from layers. These are not configurable by the end user. In ncWMS each style corresponds to a color palette. Vector layers have a further set of styles for representing vectors as arrows etc. To prevent burdening the client developer but still providing flexibility for the end user, ncWMS implements a custom mechanism in which nonstandard URL parameters are employed to modify the predefined styles and define the mapping of data values to colors from the requested palette. A `GetLegendGraphic` operation is additionally defined that allows legends to be generated to match these custom styles.
4. **Serving Metadata in JSON Format (extension)** The Capabilities document generated by the WMS-GetCapabilities request (see section 2.1) can have a large size, making it unwieldy to interpret. Therefore, in addition to the standard `GetCapabilities` request, ncWMS allows metadata to be served in JSON format (`GetMetaData`). This request is also used to pass not standard information – such as units of measure for a layer and a suggested scale range for color palettes – to the user.

ncWMS has been widely adopted by research institutes, government agencies and private industry in Europe, the US and Australia.

## 2.4 *rasdaman*: An Array Database Management System

*rasdaman* is an array DBMS, which adds storage and retrieval capabilities for massive multidimensional arrays (sensor, image, statistics data, etc.) to a standard relational DBMS. Fig. 4 illustrates this relationship. *rasdaman* sits on top of the relational database PostgreSQL, allowing clients not only to query raster data, but also to use *rasdaman* as a normal relational database. “petascope” is the component of *rasdaman* that handles geo-referenced coverage data. *rasdaman* can be accessed both via command line and through visual clients. It is based on array algebra [Bau99]. *rasdaman* has no limitation on the number of array dimensions and fully supports data even beyond spatiotemporal dimensions. *rasdaman* collections (tables) represent a binary relation, the first attribute is an object identifier and the second is the array itself. This allows foreign key references between arrays and regular relational tuples.

*rasdaman*’s declarative array query language “**rasql**” embeds itself into standard SQL by query compilation. A set of extra operations is provided for the new datatype “array”. They are based on a minimal set of algebraically defined core operators: an *i*) *array constructor*, which establishes a new array and initializes it and an *ii*) *array condensor*, which derives scalar summary information from the array (similar to SQL aggregates).

*rasdaman* is set apart from its competitors by its storage management based on chunking and tiling. Efficiency in query processing is achieved by extensive algebraic rewriting of queries and highly parallelized query operations.

An aspect that is important to the work at hand is that *rasdaman* stores arrays in column-major order [CMO] like e.g. Fortran, OpenGL and MATLAB do. There are two main methods of storing multidimensional arrays in linear order: row-major order and column-major. **Row-major order** stores arrays such that rows are stored one af-

ter the other. For example, the array  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  would be laid out in linear memory as  $[1 \ 2 \ 3 \ 4 \ 5 \ 6]$ . In contrast to that, in **column-major order**, the columns are listed in sequence, like  $[1 \ 4 \ 2 \ 5 \ 3 \ 6]$  for the above example. Treating a row-major array as a column-major array is the same as transposing it. To find the position of an array value in memory, the following formula needs to be applied:  $offset = row + column * NUMROWS$ , where *row* is the row position, *column* is the column position and *NUMROWS* is the number of rows in the array. For higher dimensions this generalizes to:

$$n_d + N_d \cdot (n_{d-1} + N_{d-1} \cdot (n_{d-2} + N_{d-2} \cdot (\dots + N_2 n_1) \dots)) = \sum_{k=1}^d \left( \prod_{\ell=k+1}^d N_\ell \right) n_k$$

where a  $d$ -dimensional array  $N_1 \times N_2 \times \dots \times N_d$  with dimensions  $N_k$  ( $k = 1, \dots, d$ ) is considered. A given element of this array is specified by a tuple  $(n_1, n_2, \dots, n_d)$  of  $d$

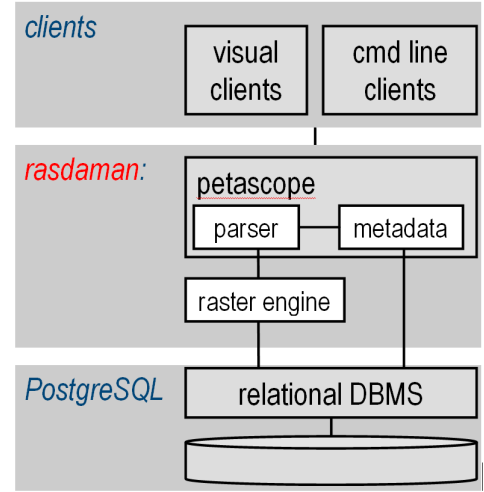


Figure 4: *rasdaman* builds on a “conventional” DBMS like PostgreSQL [PGS] (image from [RMA])

indices  $n_k \in \{0, N_k - 1\}$ .

As a consequence of these advanced techniques, *rasdaman* is significantly more efficient and scalable than other array DBMS.

## 2.5 THREDDS: Metadata Management

The **THREDDS** project [DCD<sup>+</sup>03, Car02, THRa] has developed a simple, extensible framework to provide services like metadata management and search for published datasets. It is developed and maintained by the Unidata Program Center, located in Boulder, Colorado.

Concretely, THREDDS catalogs provide a simple hierarchical structure to inventory scientific data. Each catalog contains at least a reference to a dataset and a human-readable name for it. Additional information about a dataset can be added as metadata in a number of ways. Catalogs can be used to list and obtain details for the datasets available on one particular data server, a subset from a server, the set of datasets that satisfy a query on a large body of datasets, or the datasets found about a particular topic (e.g., datasets relevant to a published paper). However many scientific datasets are still not fully described.

THREDDS catalogs may contain or reference additional information as either plain text or more structured information. Since hierarchical browsing of data is not scalable, it is convenient to allow clients to select subsets of an entire collection. The THREDDS DatasetQuery XML-format allows data servers to describe the queries clients can use to restrict the set of datasets returned. Often, very large datasets can be efficiently described by defining “product sets” of choices (i.e., choose one from column A and one from column B and so on). Dynamic data can be described using logical names (e.g., “most recent” or “latest hour”), rather than explicitly listing each dataset. Based on user selections, a query is formed which the data server resolves to a list of specific datasets.

THREDDS offers access to individual datasets by a set of different services. Two of these services are **OPeNDAP** [ODP] (local data accessibility to remote locations regardless of local storage format) and NetCDF Subsetting (see above).

Recently, the ncWMS service has been integrated with the THREDDS server such that datasets cataloged in THREDDS can now be viewed through the ncWMS interface.

More concretely, THREDDS is based on **Unidata’s Common Data Model** (CDM [CDM]). This is an abstract data model for scientific datasets. It merges the NetCDF, OPeNDAP, and HDF5 data models to create a common API for many types of scientific data. The NetCDF Java library is an implementation of CDM which can read many file formats besides NetCDF. THREDDS uses the Java NetCDF library to read any and all data. If the data is not contained in a CDM file, then THREDDS cannot access this data.

## 3 Implementation

This section contains the implementation details of integrating *rasdaman* with ncWMS and THREDDS respectively. Before my extension, *rasdaman*, THREDDS and ncWMS

each had completely separated interfaces and separate data storage methods. My implementation extends THREDDS and ncWMS with a *rasdaman* interface, allowing them to retrieve data from the *rasdaman* database directly. Fig. 5 shows this change in data flow.

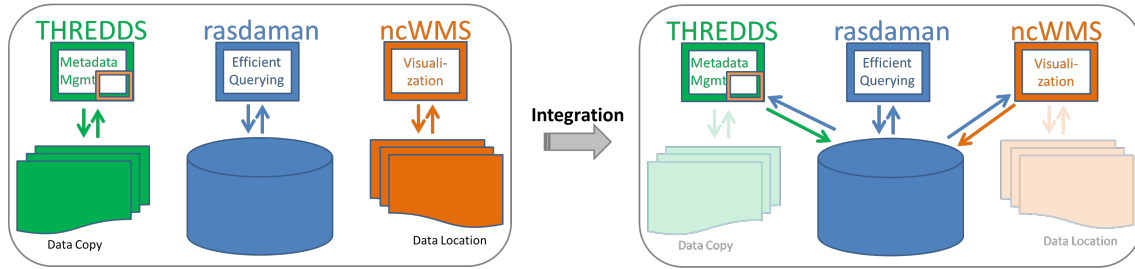


Figure 5: Dataflow before and after Integration

### 3.1 Extending ncWMS

ncWMS is implemented in 3 main layers (see Fig. 6): the HTTP layer, the presentation layer and the data readers layer.

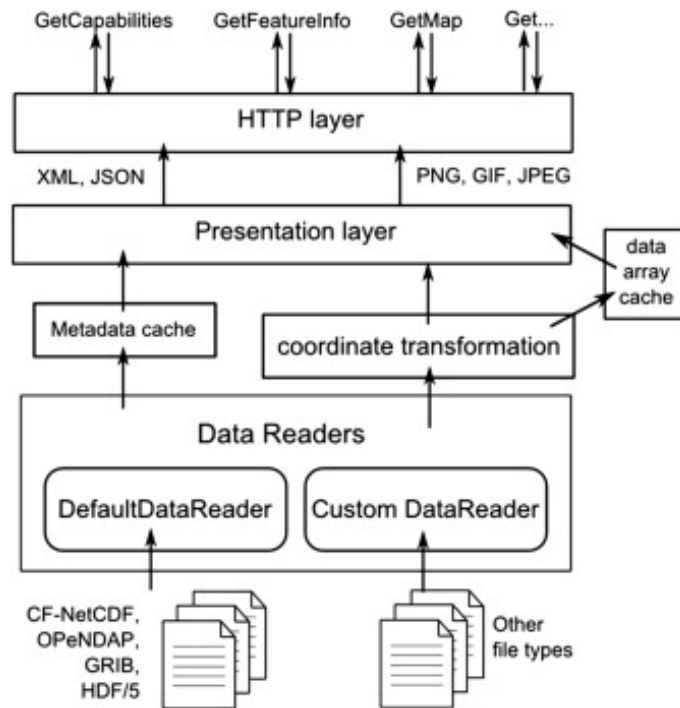


Figure 6: Outline Architecture of the ncWMS Software (image from [BGG<sup>+</sup>13])

The Data Readers are split into `DefaultDataReaders` – which read the widely used CF-NetCDF, OPeNDAP, GRIB, HDF5 data formats etc. – and `CustomDataReaders` which anyone can implement to read other, currently unsupported data formats. Not every data format can be visualized by ncWMS however, the data needs to be geo-referenced. Concretely this means that the data format can be at most 4D and needs to at least contain Latitude and Longitude axes. Each `CustomDataReader` is implemented as a Java class, which reads sections of data from a remote file. Every time a request for data is

made, the file must be read again, as the data is not saved locally.

To enable ncWMS to read data directly from *rasdaman*, I have created the *RasdamanDataReader*, a *CustomDataReader* that sends a query to the *rasdaman* database and converts the data received into the data format needed instead of opening and reading from a file each time a request is made. Each variable is encapsulated in a layer along with its metadata. This class has two main methods:

1. **getMetadata** This method reads all the metadata ncWMS requires. It returns a *CoverageMetadata* object specific to a variable, which contains the following meta-data: *i*) a variable id, title and description, *ii*) the units of the variable, *iii*) the bounding box of the data the variable contains, *iv*) a *HorizontalGrid* object (which contains information on the axes and the coordinate reference system of the variable), *v*) a *Chronology*, and *vi*) lists of time and elevation values. If any of these values are not relevant to the variable, e.g. there are no time or elevation values for a 2D latitude-longitude dataset, a null value is passed. This setup limits the data that can be passed from *rasdaman* to ncWMS to geo-referenced data with sufficient metadata. In this implementation, we restrict ourselves to coverages, provided by *rasdaman* through its WCS service. To get the necessary metadata, a *DescribeCoverage* request (see section 2.2) is sent to the *rasdaman* server and the resulting XML document is parsed.
2. **read** Once the metadata for a variable has been acquired, the actual data can be read. This method is passed a list of 2D points (Longitude, Latitude data), time, and elevation coordinates. If we are only dealing with 2D or 3D data, the time or elevation coordinate is set to -1 respectively and can be disregarded. A list of data values corresponding to the points needs to be returned (null if no value is known for that point). The algorithm for finding the list of data values is as follows:
  - (a) Send a *GetCoverage* request to the server (depending on the data only retrieve a subset or slice)
  - (b) Parse the resulting XML document for the list of data values
  - (c) Loop over the list of points
    - i. Find the position of the desired data value
    - ii. Add desired data value to the list
  - (d) Return the list of data values

Note that step 2(c)i. is not straightforward: The point we receive is of the form (lonvalue, latvalue) relative to world coordinates. However to retrieve the data, we need the indices of the matrix corresponding to these values (see Fig. 7).

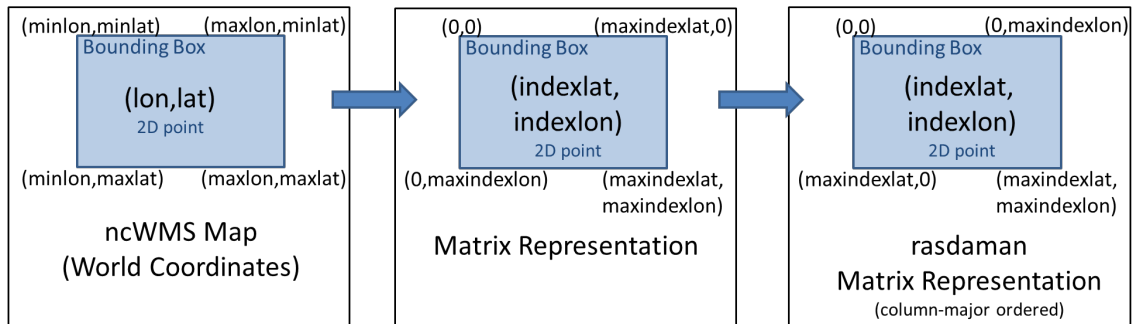


Figure 7: Transformation from Point to list position.

We shall illustrate this further using a 2D example:

Given a bounding box (5, 10, 15, 20) (corresponding to (minlat, maxlat, minlon, maxlon)) and a point (8, 18), we want to find the corresponding matrix and list position. We also know that there are 6 (nlon) points in the Lon direction and 6 (nlat) in the Lat direction.

First we need to find the matrix position corresponding to the point (8, 18). To do that we first calculate the position of the point relative to the bounding box and then convert that to absolute indices. As the resolution of the requested points may be smaller than the actual data resolution, we take the floor of the absolute index. Once we have found the indices, we can convert those into a position in the list

```
Double rellatindex = ((maxlat-lat)/((maxlat-minlat)))*(maxindexlat+1);
Double rellonindex = ((lon-minlon)/((maxlon-minlon)))*(maxindexlon+1);
int abslatindex = rellatindex.intValue();
int abslonindex = rellonindex.intValue();
return (maxindexlat+1)*abslonindex+abslatindex;
```

For our example, we obtain indices (2, 3), the position in the list is 20. The integration of *rasdaman* and ncWMS consists of one class, the *RasdamanDataReader* with about 800 lines of code.

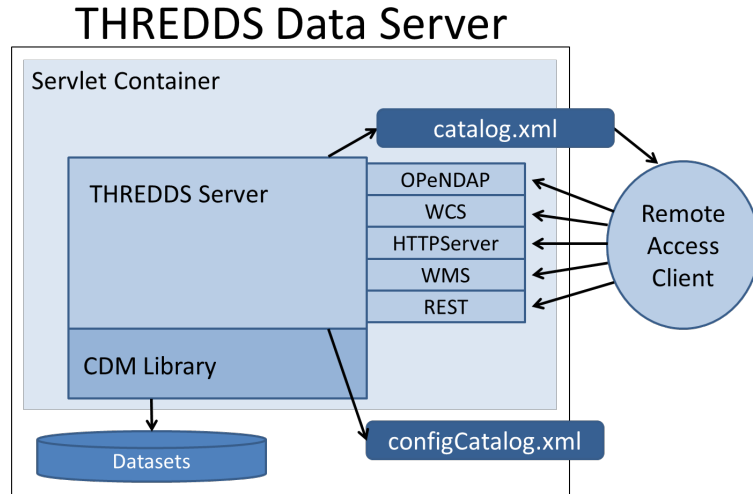


Figure 8: Outline Architecture of the THREDDS Software (image adapted from [THRb])

### 3.2 Extending THREDDS

THREDDS is based on the NetCDF Java Library: it does not store any actual datasets, but rather information on where to find them and delegates the data access to the NetCDF Java Libraries. Fig. 8 illustrates the general THREDDS architecture. The Thredds Data Server manages its catalogs defined within the `catalog.xml` file. It uses the CDM library to access stored datasets. The data is returned in different ways, depending on what service the remote client chooses.

When a dataset (of any format) is accessed, the NetCDF library opens the dataset as a `RandomAccessFile`, then passes the data to every available I/O service provider (IOSP).

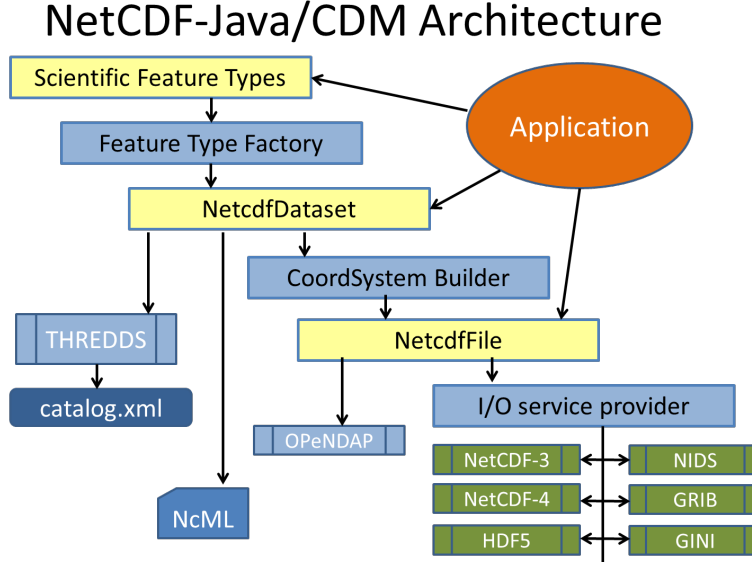


Figure 9: Data Reading Behavior of THREDDS (image from [THRb])

The first IOSP to “claim” the data is then used for all further I/O operations. The IOSP is used to parse data from the specific data format and to populate a `NetcdfFile` object, which then is used to perform further operations. Fig. 9 illustrates this. In order for an IOSP to be available to the NetCDF Java Library it needs to be registered [IR]. This can be done in one of two ways: runtime loading (which will not be covered here) and registering it within the `NetcdfFile` class. We chose to register the `RasdamanCoverageIOSP` within the `NetcdfFile` class. The IOSP has three methods that need to be implemented [ISO]:

1. `isValidFile`: When a client requests a dataset (by calling `NetcdfFile.open`), the file is opened as a `RandomAccessFile`. Each registered IOSP is then asked “is this your file?” by calling `isValidFile`. The first one that returns `true` claims it, so it is a requirement that the `isValidFile` method is fast and accurate. For the `RasdamanCoverageIOSP`, we simply send a `DescribeCoverage` request to the *rasdaman* server and make sure the response is valid. However we cannot simply pass the URL of the *rasdaman* server to THREDDS, since THREDDS requires a physical file. We solved this problem, by passing THREDDS a TXT file with the URL inside. So instead of just sending a `DescribeCoverage` request, we first parse the URL from the file and then send the request.
2. `open`: This method is used to populate the `NetcdfFile` object. A `NetcdfFile` consists of global attributes and variables (see Fig. 10). Variables have their own local attributes. An attribute is a key-value pair, an attribute name and the attribute contents. There are no restrictions on these attributes, they can be freely chosen, although one is encouraged to follow NetCDF-CF conventions [CDFc, CDFd]. Variables are used to describe each type of data in more detail. They contain attributes specific to one type of data, e.g. the units or the range of the data. It also contains information like the datatype and the dimensionality of the data. This information is used to allow the user to compose queries to the data.

To allow THREDDS to recognize that the data is geo-referenced, some extra variables and attributes are needed. For each dimension of the data, we need a so-called “coordinate variable” [CDFb] (see Fig. 11). This is a variable with the same name



**Global Attributes:**

coverage description: mean summer airtemp

**Bounding Box**

geospatial\_lat\_min: -44.525  
geospatial\_lat\_max: -8.975  
geospatial\_lon\_min: 111.975  
geospatial\_lon\_max: 156.275  
coordinate reference system: EPSG:4326

**Variables**

☐ **Lat: Array of 64 bit Reals [Lat = 0..885]**

Lat:

\_CoordinateAxisType: Lat  
units: degree

☐ **Long: Array of 64 bit Reals [Long = 0..710]**

Long:

\_CoordinateAxisType: Lon  
units: degree

☐ **pixel\_value: Grid**

Lat:  Long:

varid: 1  
units: undefined  
long\_name: pixel value  
CoordinateAxes: Lat Long  
description:

**Coordinate Reference System**

Figure 10: A Dataset and its Variables parsed from *rasdaman*

```

DDS:
Dataset {
  Dimensions
  Float64 Lat[Lat = 886];
  Float64 Long[Long = 711];
  Grid {
    ARRAY:
      Int32 pixel_value[Lat = 886][Long = 711];
    MAPS:
      Float64 Lat[Lat = 886];
      Float64 Long[Long = 711];
  } pixel_value;
} test/rasdaman2dtestdata2.txt;

```

Figure 11: A Dataset parsed from Rasdaman shown in THREDDS

as a dimension and maps values to each point of the dimension. It also contains attributes that define the axes of the coordinate reference system. Finally, we need global attributes to define, for example, a bounding box.

3. **read:** This method is used to read actual data values of a specific variable. It takes as an argument the variable and a NetCDF Section object, which contains the section of data to be read. A 1D array of data values is returned. The data values are determined as follows:
  - (a) Send a GetCoverage request to the server (depending on the dimensionality, restrict to a subset or slice)
  - (b) Parse the resulting XML Document for the list of data values
  - (c) Loop over the given points (from the Section)



- i. Convert the given indices into position of the list
- ii. Retrieve the data value
- iii. Add the data value to the array
- (d) Return the array

This algorithm is a bit simpler than the algorithm used to find data values described in 3.1, because the points in the section are already indices, not positions. Another difference is that the step size can manually be set in THREDDS, this is automatically calculated in ncWMS.

### 3.3 Test Datasets

To ensure the implementation works correctly, I used a standard 2D test set of *rasdaman*: the `mean_summer_airtemp` coverage. To test both 3D and 4D data, I used the **rasgeo** [RGU] utility provided by *rasdaman* to layer the coverage in different heights and times respectively. These coverages are stored in a local test server and can be accessed through the WCS interface of *rasdaman*.

## 4 Related Work

The goal of this work is to integrate *rasdaman* with THREDDS and ncWMS in order to create an end-to-end service allowing users to store, organise and visualize their data without having to keep three distinct copies of the same data. In the following I will present some existing solutions and how they differ from the approach described here.

Individually, all of the problems have been solved more than once: There are many versions of array DBMSs, e.g. PostGIS [PG], SciDB [SDB] and MonetDB [MDB]. *rasdaman*, however, was the first array DMBS to be released and has shaped its research domain. There are several WMS implementations besides ncWMS in use today, e.g. GeoServer [GS], ESDAC Map Viewer and GAIA [GAI]. What makes ncWMS special is that it extends WMS to also accommodate gridded environmental data. Few metadata managers like THREDDS exist, the Adaptive Metadata Manager [AMM] is an example.

However, trying to solve more than one of these problems at a time is much rarer. Some of the notable solutions are:

- The **Earthlook** [EL] project uses data from *rasdaman* to showcase several OGC standard-based services like WMS, WCS and WCPS [wcp] on Big Earth Data, in particular spatiotemporal coverages. However, when visualizing the data, the Earthlook system is severely limited in comparison to ncWMS and it can only work with one dataset at a time, while THREDDS is able to compose virtual datasets composed of multiple datasets and thus work with multiple datasets at a time.
- *rasdaman* has implemented a query front end **rView** [RV] to rasql allowing users to generate images directly from *rasdaman*. While rView as a query language is certainly more flexible than ncWMS is, it does not provide geo-semantic support, which is necessary for ncWMS.
- The **Adaptive Metadata Manager** [AMM] is a metadata manager like THREDDS. It allows users to organize metadata and trace datasets back to their source, even

back to database queries. However, it only deals with metadata – not datasets themselves – and does not support metadata updates. It has no data visualizing capabilities of its own. In short, it is an actual metadata manager, not the data *and* metadata manager my integration will provide.

- **GeoServer** [GS] is the reference implementation of the WMS and WCS standards described in the sections 2.1 and 2.2. It creates maps in a variety of output formats. GeoServer uses a free mapping library OpenLayers [OLL], making map generation easy and quick. But GeoServer only works on one dataset at a time and has no organisational capabilities like THREDDS.
- **THREDDS** has added a modular ncWMS component to visualize its data. However, this component is not a link between ncWMS and THREDDS, but a module that is now completely detached from the true ncWMS parent. Updates to ncWMS are not automatically ported to the THREDDS module. This is not ideal, since the standards on which ncWMS is built are still evolving and may make significant leaps in its functionality.

We conclude that a data-centered integration seems to be a novel and worthwhile approach.

## 5 Evaluation

I will evaluate whether or not I have achieved the goals set in section 1.2 via a cognitive walkthrough [SRP11, CWW] through the implemented demos.

Consider the following scenario: Derek’s research ship, the ”Dataminer”, is anchored off the coast of Australia. Derek and the other researchers on board are researching how the proximity to the ocean affects summer air temperatures on the mainland. To that end, they have acquired multiple datasets (some of them are stored at their institute over on the mainland). The first dataset is concerned with the mean air temperature of the current summer. The second contains air temperature at multiple elevation levels and the last deals with the air temperature measured over several days. All of this data is stored in *rasdaman* as a coverage and can be accessed via the WCS service.

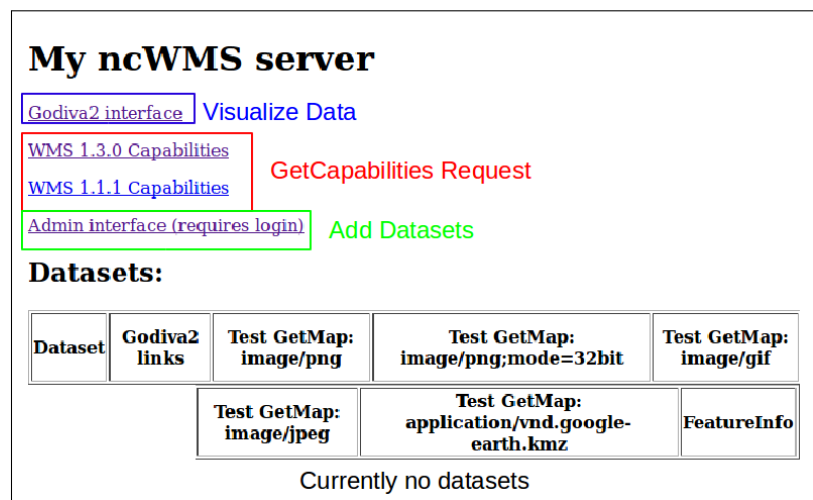


Figure 12: Derek’s ncWMS Server Main Page (fitted to a readable size)

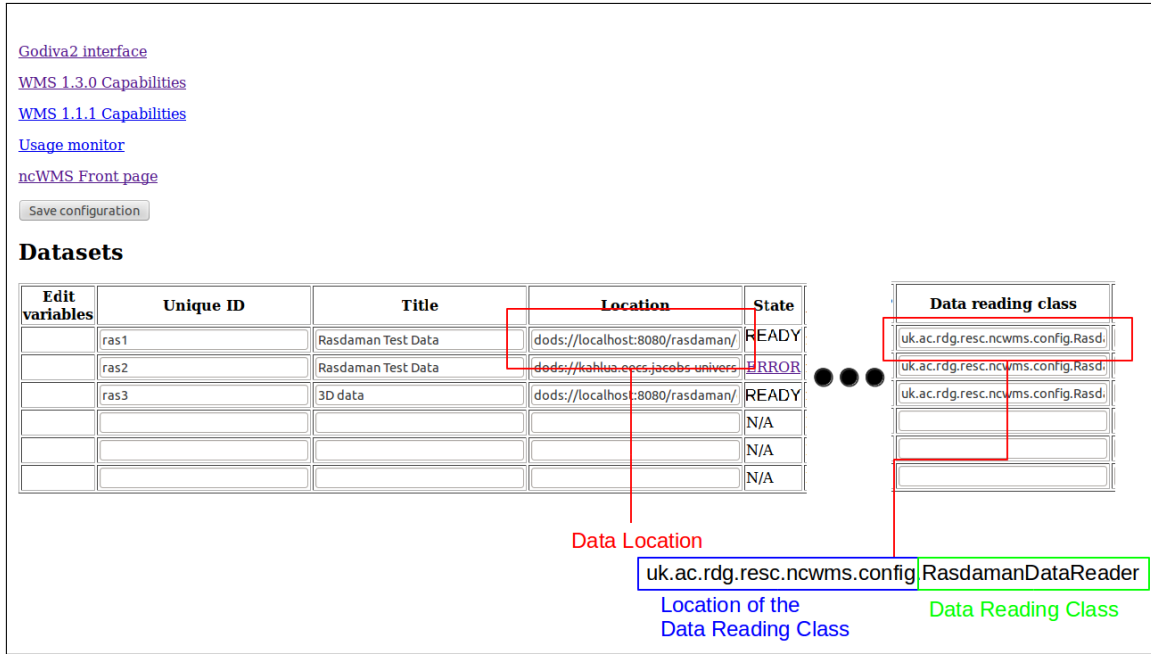


Figure 13: ncWMS admin page

## 5.1 *rasdaman* and ncWMS

Before organizing the data in THREDDS, Derek first wants a quick look whether or not the data is actually relevant to his interests. To that end he opens the main page of his ncWMS server, as shown in Fig. 12.

Since the dataset he desires is not yet present (or any for that matter), Derek navigates to the admin section and adds his dataset (*lean\_summer\_airtemp*) to the server. He needs to specify a unique id, a title, a location and most importantly a data reader for this dataset. The location is the URL to the Rasdaman Coverage and the data reader needs to be specified if the data format is not native to ncWMS. In Derek's case he specifies the newly implemented *RasdamanDatareader* (shown in Fig. 13).

Once the state of the dataset is "Ready", Derek uses the ncWMS client "Godiva2" to view the data (see Fig. 14).

Derek selects the dataset and variable he wants to view (see Fig. 15). The view of the map automatically shifts to the bounding box specified in the metadata.

Derek can customize the image several ways:

1. He can adjust the transparency of the image (see Fig. 16)
2. He can zoom in/out and navigate around the world map as he chooses (see Fig. 17).
3. He can change the color palette dynamically – the default is generated by ncWMS by sampling the data and choosing the highest and lowest values respectively as bounds (see Fig. 18).
4. For data with elevation (or time) values attached, he can choose the elevation (or time) he wants to view.

So Derek can not only visualize data taken directly from *rasdaman*, he can customize said visualization to fit his own needs.

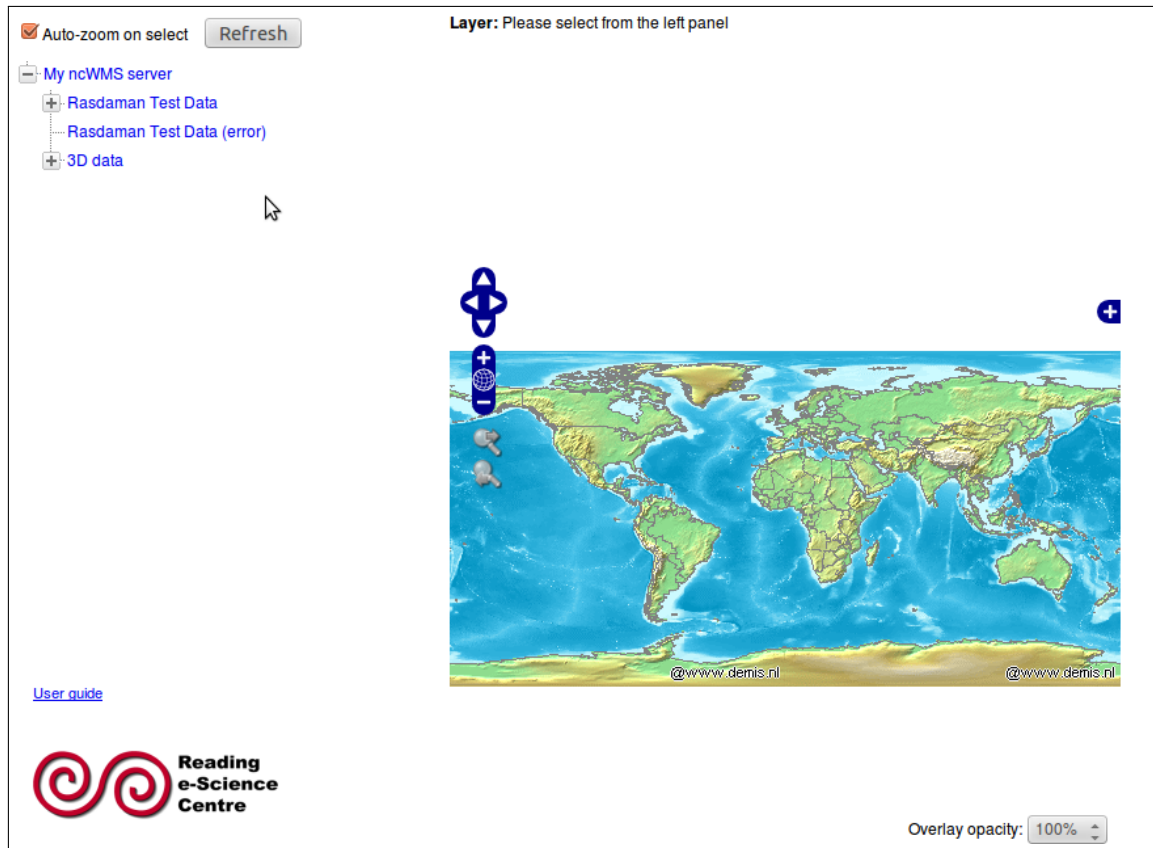


Figure 14: ncWMS GODIVA2 Visualization page, no dataset selected

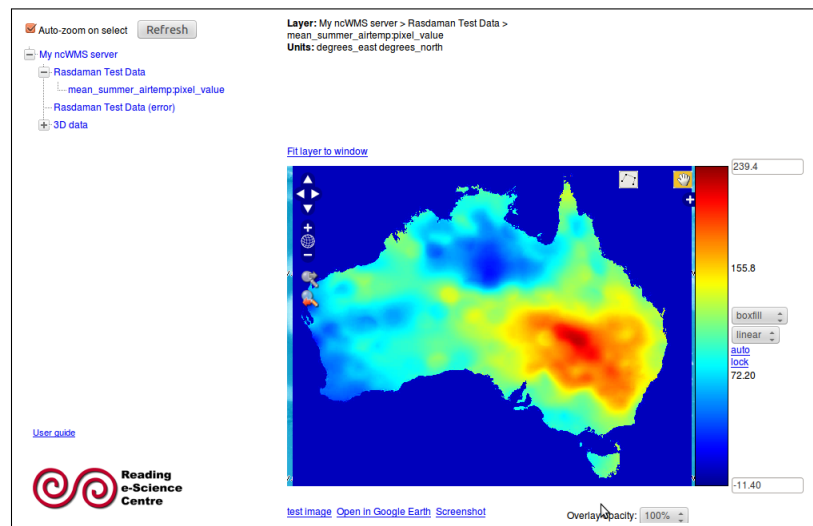


Figure 15: Initial View of mean\_summer\_airtemp

## 5.2 *rasdaman* and THREDDS

Now that Derek has determined that the gathered data is in fact suitable for his needs, he wants to organize the data. For that he needs to make/modify a THREDDS catalog. Listing 1 shows a basic catalog.

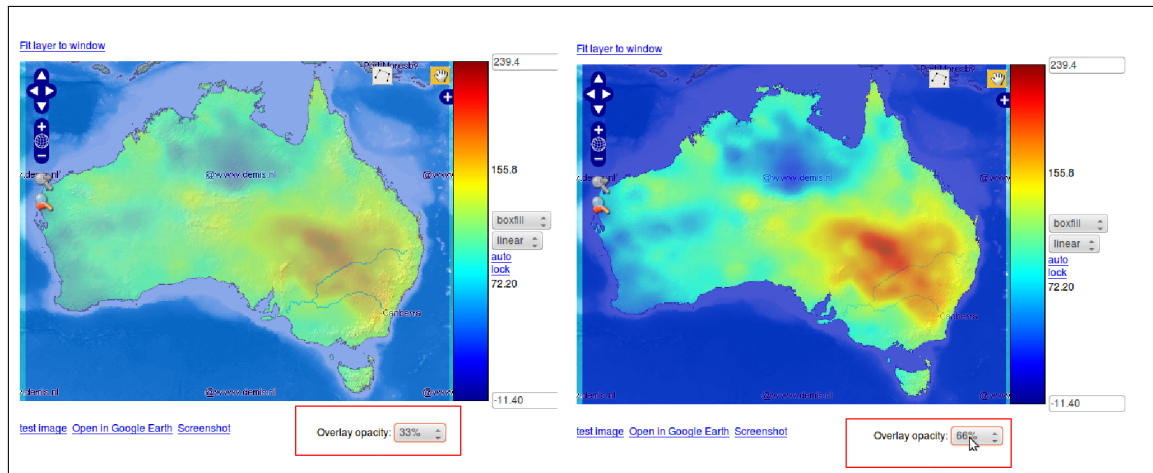


Figure 16: Different Transparency Views of mean\_summer\_airtemp

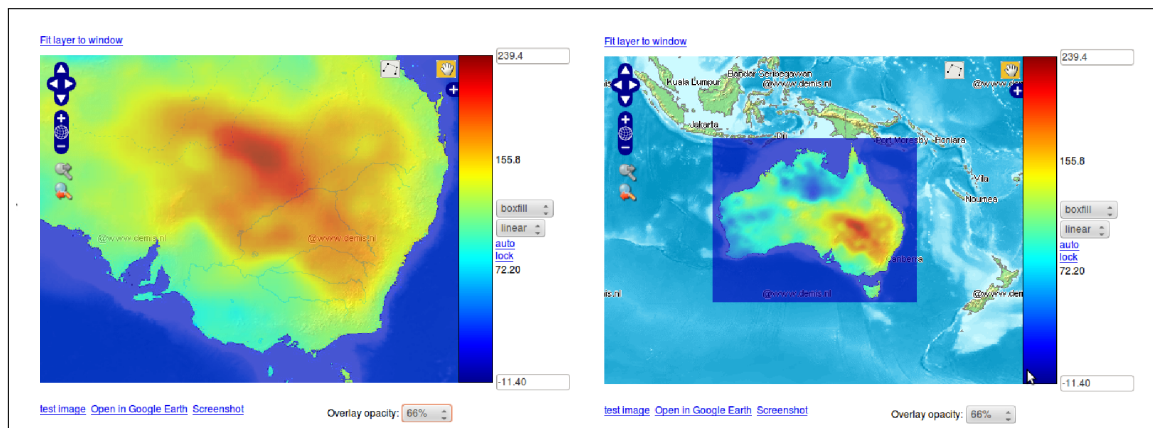


Figure 17: Zoomed in and zoomed out Views of mean\_summer\_airtemp in ncWMS

#### Listing 1: A basic THREDDS Catalog in XML Format

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog name="THREDDS_Server_Default_Catalog::You_must_change_this_to_fit_your_server!"
  xmlns="http://www.unidata.ucar.edu/namespaces/thredds/InvCatalog/v1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <service name="all" base="" serviceType="compound">
    <service name="odap" serviceType="OpenDAP" base="/thredds/dodsC/" />
    <service name="http" serviceType="HTTPServer" base="/thredds/fileServer/" />
    <!-- service name="wcs" serviceType="WCS" base="/thredds/wcs/" -->
    <service name="wms" serviceType="WMS" base="/thredds/wms/" />
    <!-- service name="ncss" serviceType="NetcdfSubset" base="/thredds/ncss/grid/" -->
  </service>
  <datasetRoot path="test" location="content/testdata/" />
  <dataset name="Test_Single_Dataset" ID="testDataset" serviceName="odap"
    urlPath="test/testData.nc" dataType="Grid" />
  <datasetScan name="Test_all_files_in_a_directory" ID="testDatasetScan" path="testAll"
    location="content/testdata">
    <metadata inherited="true">
      <serviceName>all</serviceName>
      <dataType>Grid</dataType>
    </metadata>
    <filter><include wildcard="*eta_211.nc" /></filter>
  </datasetScan>
</catalog>
```

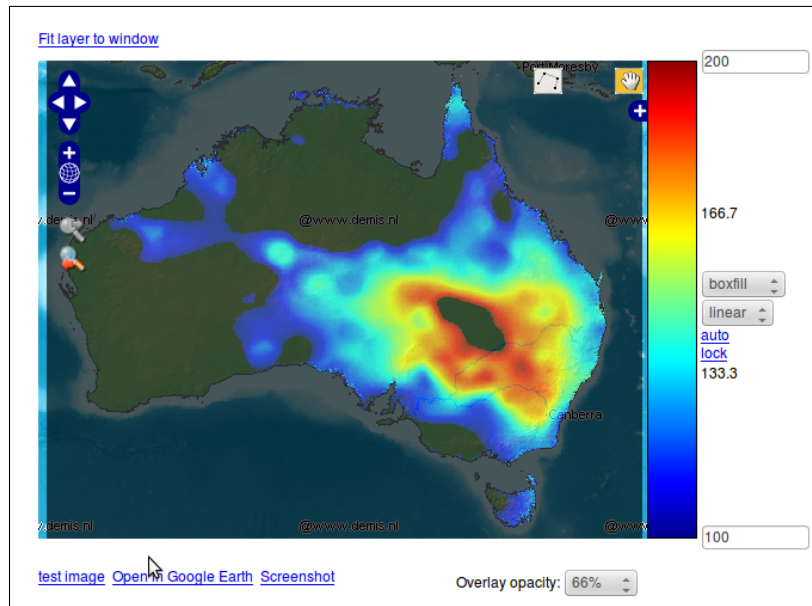


Figure 18: mean\_summer\_airtemp with Customized Palette

```
<catalogRef xlink:title="Test_Enhanced_Catalog" xlink:href="enhancedCatalog.xml" name="" />
</catalog>
```

The **service** element lists what sort of services can be applied to your data, in Derek's case, we are only interested in the OPeNDAP and the WMS services (to view and visualize the data respectively). To add a new dataset to the catalog, he needs to add a **dataset** element:

```
<dataset name="Test_Rasdaman_Single_2D_Dataset_-_mean_summer_airtemp"
  ID="testrasdaman2d2" serviceName="odap"
  urlPath="test/rasdaman2dtestdata2.txt" dataType="Grid" />
```

This element needs a name and an id in order to uniquely identify it. The **name** attribute specifies which service is to be used and the **urlPath** describes the location of the data. Derek adds all the datasets to a catalog and then goes to the main THREDDS page (Fig. 19)

There he first wants to view the actual matrix of data through the OPeNDAP service and then visualize it through the ncWMS component of THREDDS. He selects the correct dataset, which brings him to a page with further detail of the dataset (Fig. 20), where he can choose which service he wants to use to continue. Derek selects the OPeNDAP component (Fig. 21).

This service first shows Derek all the metadata available from the dataset, which he uses to further define his query. He has several options available:

- He first chooses one or more variables to query (Fig. 22)
- Then he defines the range of data within that query. He can specify the starting index, the stride and the end index (0:1:710). This means that in the data matrix, he wants only data from the starting index (0) to the end index (710), with a step size defined by the stride (1). Note that a variable can have multiple dimensions and the range data is defined for every dimension separately.





Catalog <a href="http://localhost:8080/thredds/catalog.html">http://localhost:8080/thredds/catalog.html</a>	
Dataset	Size
<a href="#">Test Single Dataset</a>	
<a href="#">Test Rasdaman Single 2D Dataset - jubcampus</a>	
<a href="#">Test Rasdaman Single 2D Dataset - mean summer airtemp</a>	
<a href="#">Test Rasdaman Single 2D Dataset - mean summer airtemp - wcs</a>	
<a href="#">Test Rasdaman Single 3D Dataset - msat 3d</a>	
<a href="#">Test Rasdaman Single 3D Dataset - msat cov rts</a>	
<a href="#">Test Single Dataset 2</a>	
 <a href="#">Test all files in a directory/</a>	
 <a href="#">Test Enhanced Catalog/</a>	
<a href="#">Initial TDS Installation at My Group</a> <a href="#">see Info</a> THREDDDS Data Server (Version 4.3.21-SNAPSHOT - 20140506.1617) <a href="#">Documentation</a>	

Figure 19: View of a THREDDDS Catalog



## Initial TDS Installation

### THREDDDS Data Server

Catalog <http://localhost:8080/thredds/>

Dataset: THREDDDS Server Default Catalog - mean\_summer\_airtemp

- Data type: GRID
- ID: testrasdaman2d2

**Access:**

1. **OPENDAP:** </thredds/dodsC/test/rasdaman2dtestdata2.txt>  
**WMS:** </thredds/wms/test/rasdaman2dtestdata2.txt>

**Viewers:**

- [Godiva2](#) (browser-based)
- [NetCDF-Java ToolsUI](#) (webstart)
- [Integrated Data Viewer \(IDV\)](#) (webstart)

Figure 20: View of a detail dataset page

Once he has defined his query, THREDDDS actually retrieves the data (Fig. 23).

Now Derek has seen the data and can draw some conclusions. He needs to present these to his clients, but they certainly won't be able to understand these columns of numbers. Therefore he now also wants to view the same data through the ncWMS component of THREDDDS. For that he navigates back to the detail page and chooses the Godiva2 service (Fig. 24). This allows him to visualize and customize his data, just as described in section 5.1.

Derek has now organized and visualized data directly from *rasdaman*, without having to do tedious copy and paste work at all. This constitutes the end-to-end service envisioned in section 1.2.

## OPeNDAP Dataset Access Form

---

Tested on Netscape 4.61 and Internet Explorer 5.00.

---

**Action:**

**Data URL:**

---

**Global Attributes:**

coverage\_title: mean\_summer\_airtemp  
 coverage\_description: mean\_summer\_airtemp  
 geospatial\_lat\_min: -44.525  
 geospatial\_lat\_max: -8.975  
 geospatial\_lon\_min: 111.975  
 geospatial\_lon\_max: 156.275

---

**Variables:**

☐ **Lat: Array of 64 bit Reals [Lat = 0..885]**

Lat:   
 \_CoordinateAxisType: Lat  
 units: degree

☐ **Long: Array of 64 bit Reals [Long = 0..710]**

Long:   
 \_CoordinateAxisType: Lon  
 units: degree

☐ **pixel\_value: Grid**

Lat:  Long:   
 varid: 1  
 units: undefined  
 long\_name: pixel\_value  
 \_CoordinateAxes: Lat Long  
 description:

---

*For questions or comments about this dataset, contact the administrator of this server [Support] at: [support@my.group](mailto:support@my.group)*

*For questions or comments about OPeNDAP, email OPeNDAP support at: [support@opendap.org](mailto:support@opendap.org)*

---

**DDS:**

```

Dataset {
  Float64 Lat[Lat = 886];
  Float64 Long[Long = 711];
  Grid {
    ARRAY:
      Int32 pixel_value[Lat = 886][Long = 711];
    MAPS:
      Float64 Lat[Lat = 886];
      Float64 Long[Long = 711];
  } pixel_value;
} test/rasdaman2dtestdata2.txt;
  
```

Figure 21: Metadata view of OPeNDAP service

☒ **pixel\_value: Grid**

Lat:  Long:

varid: 1  
 units: undefined  
 long\_name: pixel\_value  
 \_CoordinateAxes: Lat Long  
 description:

Figure 22: A Selected Variable, with Query Options





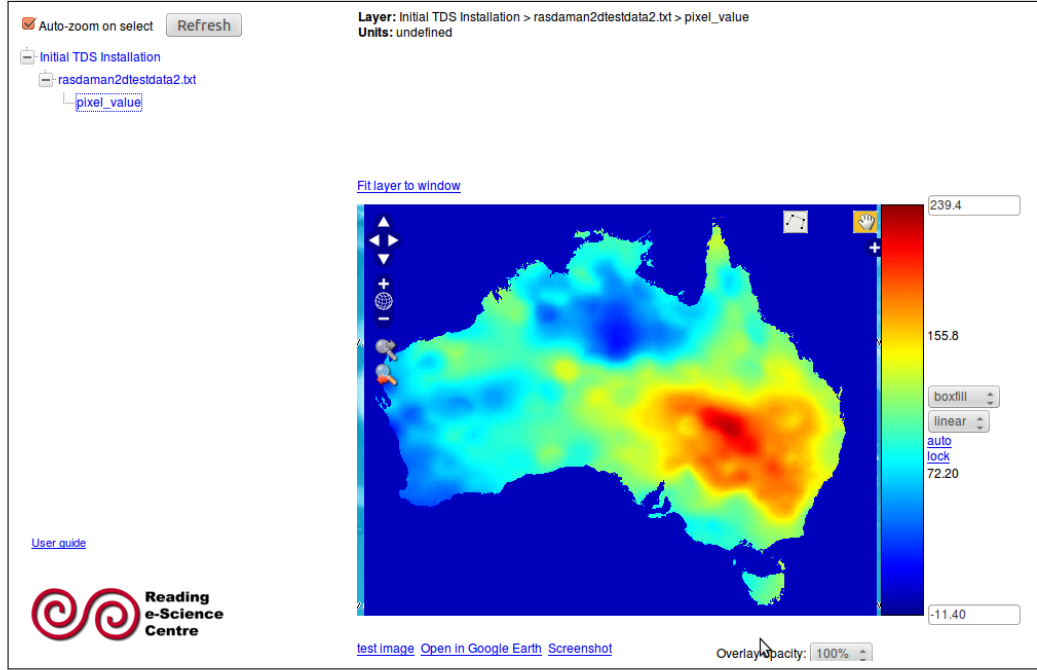


Figure 24: The Result of an ncWMS Visualization of Data from THREDDS

THREDDS, but any program using the Java NetCDF Libraries to access data directly from *rasdaman*.

We have evaluated the feasibility and usability of our answers with a cognitive walkthrough of our running example in Section 5.

In the course of the implementation we learned that the architectures focused on in this thesis are especially well-suited to establishing interfaces, since all three implement standardized data formats and communication protocols. This helps clearly define the connecting features, where interface links can be forged. Another positive aspect of these standards is that all data must be supplied with metadata, which is vital for many services, but often overlooked in the process of gathering data. Standardizing interfaces also furthers software evolution since it allows applications to specialize, optimize the services they offer and compete on functionality. *rasdaman* for example is very good at storing and querying array data.

However, these standards are still in development, causing version conflicts. For instance, the WMS standard has switched its latitude-longitude axis order to longitude-latitude order, which caused problems in my implementation initially. Some applications like ncWMS have already implemented this new standard but other dependent applications have not, making some visualizations more difficult.

My work has accomplished the basic link between *rasdaman*, THREDDS and ncWMS, but it is far from perfect. Steps to further enhance the result include:

1. Extending my CustomDataReader in ncWMS and the IOSP in THREDDS to receive TIFF-encoded data for greater efficiency.
2. Adding WCPS support, instead of just WCS support to allow even more customizable queries. WCPS [wcp] is the Web Coverage Processing Service and defines an actual query language for filtering and processing Coverages. This query language

is embedded into WCS, and thus allows for much more flexible querying.

3. Changing the ncWMS module of THREDDS into a link between the “parent ncWMS” and THREDDS to allow further progress to be reflected.
4. Adding an “Add dataset” form to the GUI of THREDDS, so users do not have to change the actual catalog file in XML format.

I hope that this thesis goes a step in establishing *rasdaman* as the standard array DBMS for current and novel geo-information systems.

**Acknowledgements** I would like to thank Prof. Dr. Peter Baumann for the guidance and support throughout this guided research project. Also I would like to thank Jon Blower from the University of Reading for the provided help. I appreciated very much the support from Piero Campalani and Constantin Jucovski. Finally, for seeing me through to the end I thank my parents Andrea and Michael Kohlhase.

## References

- [AMM] Adaptive Metadata Manager. homepage at <http://www.adaptive.com/products/metadata-manager/>; visited: 4.12.2013.
- [Bau99] P. Baumann. A database array algebra for spatio-temporal data and beyond. In *Proc. NGITS99*, number 1649 in LNCS, pages 76–93. Springer, 1999.
- [BGG<sup>+</sup>13] J.D. Blower, A.L. Gemmell, G.H. Griffiths, K. Haines, A. Santokhee, and X. Yang. A web map service implementation for the visualization of multidimensional gridded environmental data. *Environmental Modelling & Software*, 47(0):218 – 224, 2013.
- [Car02] John Caron. THREDDS: A geophysical data/metadata framework. In *Preprints, 18th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*. American Meteorological Society, 2002.
- [CDFa] netCDF. homepage at <http://www.unidata.ucar.edu/software/netcdf/>; visited: 4.12.2013.
- [CDFb] Netcdf Best Practices. homepage at <https://www.unidata.ucar.edu/software/netcdf/docs/BestPractices.html>; visited: 27.4.2014.
- [CDFc] Netcdf Conventions. homepage at <https://www.unidata.ucar.edu/software/netcdf/conventions.html>; visited: 27.4.2014.
- [CDFd] NODC Netcdf Templates. homepage at <http://www.nodc.noaa.gov/data/formats/netcdf/>; visited: 27.4.2014.
- [CDM] Unidata Common Data Model. instructions at <http://www.unidata.ucar.edu/software/thredds/current/netcdf-java/CDM/>; visited: 1.12.2013.
- [CFM] Climate and Forecast Metadata Convention. homepage at <http://cf-pcmdi.llnl.gov/>; visited: 4.12.2013.
- [CMO] Column Major Ordered Data. homepage at [http://en.wikipedia.org/wiki/Row-major\\_order](http://en.wikipedia.org/wiki/Row-major_order); visited: 27.4.2014.
- [CWW] Cognitive Walkthrough Wikipedia Entry. at [http://en.wikipedia.org/wiki/Cognitive\\_walkthrough](http://en.wikipedia.org/wiki/Cognitive_walkthrough); visited: 10.5.2014.
- [DCD<sup>+</sup>03] Ethan R. Davis, J. Caron, B. Domenico, R. Kambic, and S. Nativi. THREDDS: publishing, cataloging, describing, and discovering scientific datasets. In *19th Conference on IIPS*. American Meteorological Society, 2003.
- [EL] Earthlook. homepage at <http://earthlook.org//index.html>; visited: 4.12.2013.
- [GAI] GAIA, ESDAC. at <http://eusoils.jrc.ec.europa.eu/wms/wms.htm>; visited: 4.12.2013.
- [GS] GeoServer. at <http://geoserver.org/display/GEOS/Welcome>; visited: 10.5.2014.
- [GSD] Geo Spatial Data. infor at <http://www.rasdaman.com/Images/earthcube.gif>; visited: 4.12.2013.

- [IR] IOSP Registering. project homepage at <http://www.unidata.ucar.edu/software/thredds/current/netcdf-java/reference/RuntimeLoading.html#ServiceRegistry>; visited: 4.05.2014.
- [ISO] IOSP Overview. project homepage at <http://www.unidata.ucar.edu/software/thredds/current/netcdf-java/tutorial/IOSPOverview.html>; visited: 4.05.2014.
- [MDB] MonetDB. at <https://www.monetdb.org/Documentation/Extensions/GIS>; visited: 10.5.2014.
- [NCW] ncWMS. project homepage at <http://www.resc.rdg.ac.uk/trac/ncWMS/wiki>; visited: 1.12.2013.
- [ODP] OPeNDAP. homepage at <http://www.opendap.org/>; visited: 4.12.2013.
- [OGC] Open Geospatial Web Standards. project homepage at <http://www.opengeospatial.org/standards>; visited: 10.05.2014.
- [OLL] OpenLayers library. at <http://openlayers.org/>; visited: 10.5.2014.
- [PG] POSTGIS. at <http://postgis.net/>; visited: 10.5.2014.
- [PGS] PostGreSQL. homepage at <http://www.postgresql.org/>; visited: 4.12.2013.
- [RDM] RasDaMan. project homepage at <http://www.RasDaMan.org/>; visited: 1.12.2013.
- [RGU] rasgeo utility. at <http://www.rasdaman.org/wiki/RasgeoUserGuide>; visited: 10.5.2014.
- [RMA] Rasdaman Architecture. infor at <http://wiki.ieee-earth.org/@api/deki/files/386/=image15.png?size=>; visited: 4.12.2013.
- [RV] rView. at [http://www.faculty.jacobs-university.de/pbaumann/iu-bremen.de\\_pbaumann/Courses/InformationArchitecture/RasDaMan-doc/pdf/inst-guide.pdf](http://www.faculty.jacobs-university.de/pbaumann/iu-bremen.de_pbaumann/Courses/InformationArchitecture/RasDaMan-doc/pdf/inst-guide.pdf); visited: 4.12.2013.
- [SDB] SciDB. at <http://www.scidb.org/>; visited: 10.5.2014.
- [SRP11] Helen Sharp, Yvonne Rogers, and Jenny Preece. *Interaction design: beyond human-computer interaction*. John Wiley & Sons, 3 edition, 2011.
- [THRa] THREDDDS. project homepage at <http://www.unidata.ucar.edu/software/thredds/current/tds/TDS.html>; visited: 1.12.2013.
- [THRb] THREDDDS Dataserver Talk. infor at [http://www.youtube.com/watch?v=zt\\_s64GqewM](http://www.youtube.com/watch?v=zt_s64GqewM); visited: 4.12.2013.
- [wcp] WCPS standard. at <http://www.opengeospatial.org/standards/wcps>; visited: 10.5.2014.
- [WCSa] WCS. project homepage at <http://www.opengeospatial.org/standards/wcs>; visited: 3.05.2014.
- [WCSb] WCS specification. project homepage at <https://portal.opengeospatial.org/files/09-110r4>; visited: 3.05.2014.

- [wcsc] WCS Wikipedia Entry. homepage at [http://en.wikipedia.org/wiki/Web\\_Coverage\\_Service](http://en.wikipedia.org/wiki/Web_Coverage_Service); visited: 10.5.2014.
- [WMSa] WMS standard. homepage at <http://www.opengeospatial.org/standards/wms>; visited: 4.12.2013.
- [wmsb] WMS Wikipedia Entry. homepage at [http://en.wikipedia.org/wiki/Web\\_Map\\_Service](http://en.wikipedia.org/wiki/Web_Map_Service); visited: 10.5.2014.