**Siemens AG**

National Geothermal Data System Software Architecture Document (SAD)

Ver sion: V1.0

### CONT ENT OWNER: Chri stoph Kuhm uench

|  |  |  |  |
| --- | --- | --- | --- |
| **Version:** | **Date:** | **Author:** | **Detail s** |
| 0.1 | 05/ 20/ 2013 | Kuhmuench | Creation of Template |
| 0.3 | 07/ 16/ 2013 | Kuhmuench/ Bruschi | Release for Review b y P. Bruschi |
| 0.4 | 07/ 22/ 2013 | Bruschi | Review contents. |
| 0.5 | 07/ 25/ 2013 | Kuhmuench | Integrated comments |
| 0.6 | 07/ 30/ 2013 | Kuhmuench | Completing several sections and tables. |
| 0.7 | 07/30/2013 | Bruschi | Reviewed complete document |
| 1.0 | 08/20/2013 | Kuhmuench | Finishing Touches |

# Table of Contents

1. [Preface 3](#_TOC_250062)
   1. [Pur pose and Audience 3](#_TOC_250061)
   2. [Responsibili ti es (Stakeholders) 4](#_TOC_250060)
   3. [D ocum ent Roadm ap 5](#_TOC_250059)
2. [System Scope and Back ground 6](#_TOC_250058)
   1. [Probl em Background 6](#_TOC_250057)
      1. [Goals and Context 7](#_TOC_250056)
      2. [Sig ni ficant Dr iving Requirements 10](#_TOC_250055)
   2. [Fur ther Reading on Requir em ents and Functionali ty 12](#_TOC_250054)
   3. [Solution Back ground 15](#_TOC_250053)
      1. [Architectural App roaches 15](#_TOC_250052)
      2. [Analysis Results 15](#_TOC_250051)
   4. [Product Line Reuse Considerati ons 16](#_TOC_250050)
3. [Use Case Scenari os 17](#_TOC_250049)
4. [M ain Decisions 20](#_TOC_250048)
   1. [One Software for both Node-i n-a-Box and Central Node 20](#_TOC_250047)
   2. [Operating System Support 20](#_TOC_250046)
   3. [Base Backend Technol ogies 20](#_TOC_250045)
   4. [Base Frontend Technologi es 21](#_TOC_250044)
5. [Hi gh-l evel Context Vi ews 22](#_TOC_250043)
   1. [System Context Vi ews 22](#_TOC_250042)
   2. [Logi cal Vi ews 23](#_TOC_250041)
      1. [Domain Model 23](#_TOC_250040)
      2. [Hig h-level Components 26](#_TOC_250039)
         1. [CKAN 26](#_TOC_250038)
         2. [SOLR 27](#_TOC_250037)
         3. [Geoser ver 27](#_TOC_250036)
         4. [Postgres/ PostGIS 27](#_TOC_250035)
         5. [CKAN Data-store and Data-storer Extension 27](#_TOC_250034)
         6. [OGC Controll er Extension 28](#_TOC_250033)
         7. [Upload Management 28](#_TOC_250032)
         8. [Bulk Upload Management 28](#_TOC_250031)
         9. [Content Model Management 28](#_TOC_250030)
         10. [CKAN-Spatial Extension 29](#_TOC_250029)
         11. [CKAN Har vesting Extension 29](#_TOC_250028)
         12. [CKAN Full -text Indexing Extension 29](#_TOC_250027)
         13. [NGDS UI 29](#_TOC_250026)
         14. [CKAN Map Extension 29](#_TOC_250025)
      3. [Layered View 30](#_TOC_250024)
         1. [Pyl ons 30](#_TOC_250023)
         2. [Sql Alchemy 30](#_TOC_250022)
         3. [Jinja2 31](#_TOC_250021)
         4. [Vir tualenv 31](#_TOC_250020)
         5. [PIP 31](#_TOC_250019)
         6. CKAN, CKAN RESTful API, and CKAN Extension Interface31
         7. [CKAN Extensions 31](#_TOC_250018)
   3. [Developm ent Views 34](#_TOC_250017)
      1. [Component View of CKAN 34](#_TOC_250016)
      2. [Github Code Overview 36](#_TOC_250015)
   4. [System Dynam i cs Vi ews 38](#_TOC_250014)
      1. [Use Case: “ Up load unstr uctured Dataset Resource” 38](#_TOC_250013)
      2. [Use Case: “ Up load Str uctured Dataset Resource” 40](#_TOC_250012)
      3. [Use Case: “ Publi sh Str uctured Resource as OGC Ser vices” 42](#_TOC_250011)
      4. [Use Case: “ Map Search” 43](#_TOC_250010)

ii last saved: Tuesday, August 20, 2013

* + 1. [Use Case: “ Keyword Lib rar y Search” 44](#_TOC_250009)
    2. [Use Case: “ Upl oad Shapefile” 44](#_TOC_250008)
    3. [Use Case: “ Har vest Metadata” 45](#_TOC_250007)
    4. [Use Case “ Preview WFS WCS Resources” 47](#_TOC_250006)
  1. [Process Views 47](#_TOC_250005)
  2. [Physi cal Vi ews 49](#_TOC_250004)

1. [Referenced M ateri als 51](#_TOC_250003)
2. [Di rector y 52](#_TOC_250002)
   1. [Gl ossar y 52](#_TOC_250001)
   2. [Acronym Li st 53](#_TOC_250000)

# List of Figures

Fig ure 1: 4+ 1 View Model for Softwar e Architectur es 3

Fig ure 2: NGDS Stakehold ers 4

Fig ur e 3: Basic Goals of NGDS 8

Fig ur e 4: NGDS is a grid of repositori es 9

Fig ure 5: NGDS System Context View 23

Fig ur e 6: NGDS Domain Model as a Class Diagram 24

Fig ur e 7: NGDS Domain Model of Resources as a Class Diagram 25

Fig ure 8: NGDS Hig h-level Components 26

Fig ure 9: NGDS Layered View 30

Fig ure 10: Pylons Base Libr ari es 35

Fig ure 11: CKAN Component View 36

Fig ure 12: Content of the base dir ectory of the repository ckanext-ngd s 38

Fig ure 13: MSC for Use Case "Upl oad Dataset with Unstructured Resource" 39

Fig ure 14: MSC for Use Case "Upl oad Dataset with Str uctured Resour ce" 41

Fig ure 15: MSC for Use Case " Publi sh structured r esource as OGC Services" 42

Fig ur e 16: MSC for Use Case "Map Search" 43

Fig ure 17: MSC for Use Case "Keyword Search” 44

Fig ure 18: MSC for Use Case "Shapefil e Upload" 45

Fig ur e 19: MSC for Use Case "Harvest Metadata" 46

Fig ure 20: Use Case “ Pr eview WFS WCS Resources” 47

Fig ure 21: NGDS scaled hori zontally 49

Fig ur e 22: NGDS scaled ver tically 50

# List of Tables

Tab le 2-1: Architecturally sig ni ficant functional features 11

Tab le 2-2: Architecturally sig ni ficant quali ty featur es 12

Tabl e 2-3: NGDS Feature Gr oups 14

Tab le 2-4: Rating of Framework Candid ates 16

Tab le 3-1: Use Case Scenari os 19

Tabl e 5-1: Exi sting CKAN Extensions used in NGDS 33

Tabl e 5-2: New CKAN Extensions developed by the NGDS team 34

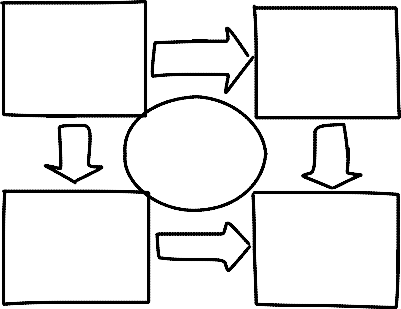
Tab le 5-3: NGDS Repositori es on http:// github .com 37

Tabl e 5-4: NGDS Processes 48

## Preface

This Software Architecture Document (SAD) documents the software architecture of the National Geothermal Data System (NGDS). The document’s outline is inspired by Kruchten’s 4+1 View Model for Software Architecture [KRUCHTEN].

**class 4+1**



Logical View

Development View

Scenarios

Process View Physical View

*Figure 1: 4+1 View Model for Software Architectures*

Kruchten’s model is chosen because it is widely accepted and allows a very fine-grained tailoring to the specific needs of a project. Furthermore the model is reasonably lightweight.

The description of the NGDS architecture utilizes the document template provided by the Software Engineering Institute (SEI) [SEI]. However, this template is very complex and goes beyond the needs of a project of the size of NGDS. Also, it is hard to read because it contains many cross-references between the sections of the document and a deep hierarchy level.

Therefore the two approaches are melded together by thinning out the SEI document template and replaced sections with ideas from the 4+1 View Model.

The theme of this architecture document is to strike a balance between lightweight documentation and structured and detailed insight.

### Purpose and Audience

This document is intended for a technical audience who need to understand the concepts and the reasoning behind the NGDS architecture. Targeted audience includes:

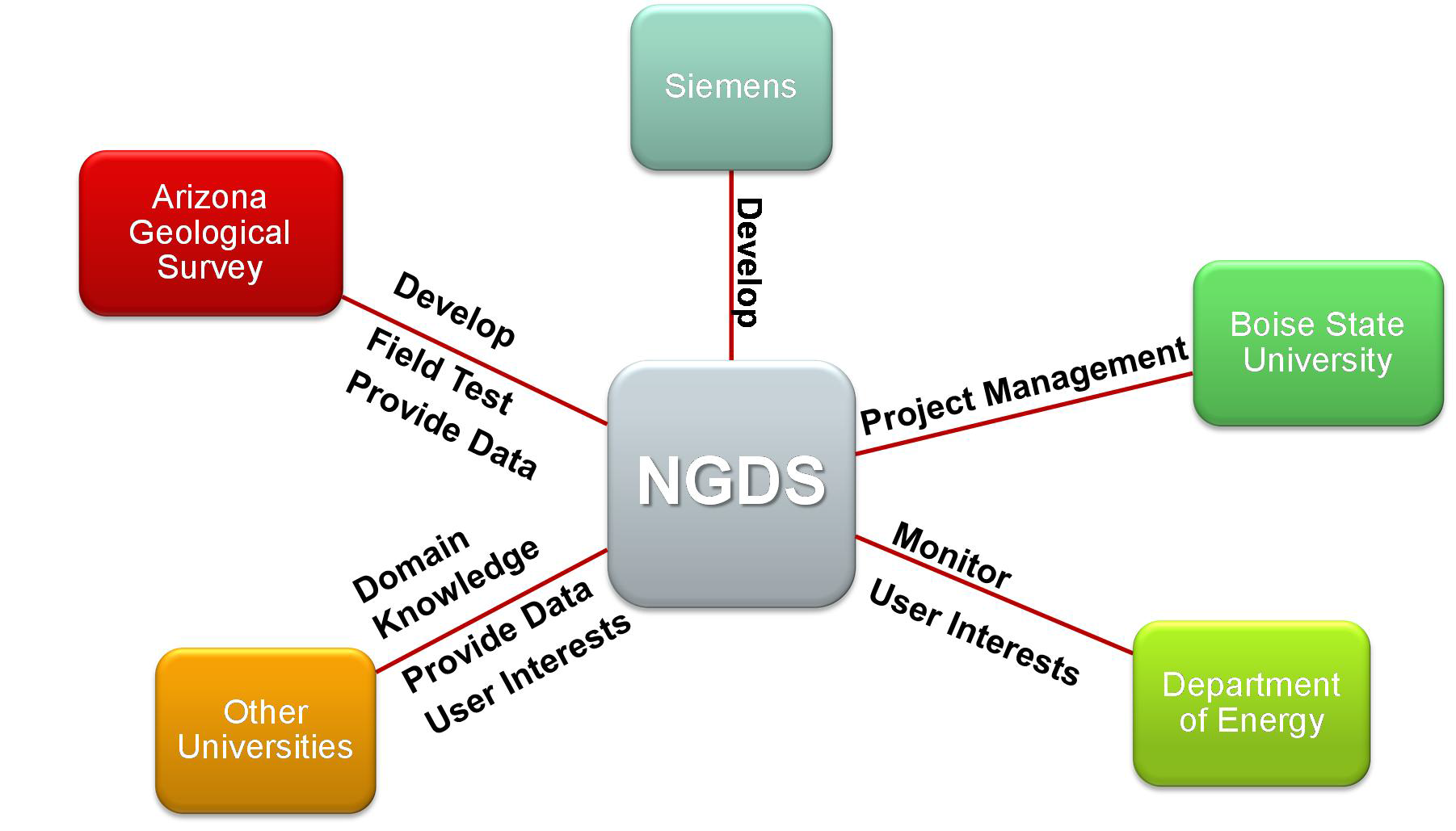
* + - Software Architects
    - Software Developers
    - DoE Monitors

The purpose of the document is to clarify for the audience how NGDS works, which base components are used and what the main technical decisions were made during the

design and implementation of NGDS. This helps to fulfill one of the main goals of NGDS, which is to provide a basis for a sustainable open source software project that is attractive for an open source team to maintain. With this documentation future developers will be able to quickly understand the system and become productive. The NGDS documentation also provides the basis for a future open source organization to take over architectural oversight of the software.

### Responsibilities (Stakeholders)

Figure 2 depicts the stakeholder organizations of NGDS considered in the development of the architecture described by this SAD. This section lists the concerns that each stakeholder organization has that can be addressed by the information in this SAD.



*Figure 2: NGDS Stakeholders*

The Department of Energy is the project sponsor of NGDS. The DOE represents the interests of all NGDS end users including commercial/ industrial/ financial end users. For the DOE it is important to have a selection of data in the system that is of interest to the various user groups and that it is made available in a simple and easy to understand way. It is also important to the DOE to ensure that the product is sustainable and lives on after the project funding is exhausted.

Boise State University has the overall project lead and is responsible to ensure that the deliverables are created and that all project partners are cooperating harmoniously.

Boise’s main interest is that the project creates a useful and attractive output that will be useful to the NGDS end users.

Siemens’ role is to understand the requirements of NGDS, implement the system according to the requirements, and provide the documentation and installation tools. Siemens interest is to provide a functional and valuable solution that provides the expected features. Siemens’ role is to research powerful technical solutions and provide an effective and efficient system that meets the customer’s needs.

Arizona Geological Survey (AZGS) also contributes to the implementation of the system, and provides Siemens with domain knowledge, executes the field tests, and is a source of some of the NGDS data. AZGS is a candidate to provide NGDS as both a Node-in-a- Box and possibly act as a Central Node. Thus, AZGS is interested in the relevant installation and data upload features. AZGS wants to profit from being a core host of geothermal data. AZGS is also a candidate to play a major role in long-term maintenance of NGDS.

Other universities are providing domain knowhow, data and end user requirements. Universities will make use of the NGDS software as a Node-in-a-Box and participate in the NGDS grid of shared geothermal information. Their interest is to increase the quality and quantity of their research activities by sharing their NGDS data with other universities and organizations.

### Document Roadmap

This document outlines the architecture of NGDS and is structured in the following way:

* + - Section 2 provides a brief overview of the system scope and background.
    - Section 3 describes the most important use cases and scenarios.
    - Section 4 highlights the key architectural decisions that were made during the implementation of NGDS.
    - Section 5 contains five views of the NGDS Design (logical view, development view, process view, and physical view).

## System Scope and Background

The requirements of NGDS are discussed in detail in the requirements document [REQ2.7]. This section discusses the most important aspects of those requirements that are important for understanding what the NGDS does and its architecture.

### Problem Background

Creating (electrical) energy from geothermal resources is a business with a high level of risk. Investments before exploitation can take place are high and it takes many years before they become profitable; however, geothermal energy is a renewable form of energy and therefore interesting as a long term energy resource.

It is the U.S. Department of Energy (or DOE) vision to discover and enable exploitation geothermal energy sources. The DOE and other organizations are funding a variety of research activities around this vision. So far these research activities focus on either collecting geothermal data or presenting and analyzing such data.

###### Existing Data Collection Activities:

Prior data collection activities focused on the creation of various repositories of geothermal data. For example:

* + - DOE-GDR – Geothermal Data Repository (htt p s:/ / g dr.op enei .or g),
    - SMU – Southern Methodist University (htt p :/ / g eothe r mal .smu.ed u),
    - EGI – Energy and Geosciences Institute (htt p :/ / egi .utah. ed u/ ),
    - USGS – U.S. Geological Survey

(htt p :/ / energy.usg s.g ov/ Other Ene r gy / Geothe r mal .aspx),

* + - and AASG – Association of American State Geologists (htt p :/ / ww w.stateg eother mald ata.or g / ).

These repositories are used to collect structured data (e.g. well headers or heat flow data expressed in well formed tables) and unstructured data (mainly publications as well as other documents – which have content as varied as text, photos and hand-written text, that are usually not represented in well-known tabular formats). Due to these initiatives, a large amount of data has been made available digitally including structured data aggregated in datasets which are exposed as Web Feature Services [WFS], Web Map Services [WMS] and Web Coverage Services [WCS] while unstructured data is made available for download. These repositories are all based on individual software systems that all comply with standardized protocols (WFS, WMS, and WCS).

Moreover, each repository exposes a catalog that allows for searching within the data of the repository. The catalog has a metadata entry for each structured dataset (i.e. aggregated structured data). The metadata entry describes the content of the dataset, such as the type of data (e.g. heat flow, well log, etc.), its origin, and the geographic region covered by the dataset. The details of the metadata entry depend, at least partially, on the type of data. In the case of unstructured data, the metadata entry comprises the results of a keyword indexing service.

The catalog is made accessible via the Catalog Service for the Web (CSW) for all users without any password protection. The publication of datasets in NGDS, however, is restricted to authorized users.

###### Existing Presentation and Analysis of Data Activities:

Data analysis activities focus on the development of tools (such as the Geothermal Prospector http :// map s.nrel .g ov/ g t\_pr osp ect or) that allow for analyzing and visualizing data. There are also commercial tools for data analysis available.

###### How is NGDS going to contribute to this landscape of research project results?

NGDS fills the gap between data collection activities and data analysis activities in existing DOE funded projects. The most important new feature is NGDS’s ability to harvest the catalogs of all existing geothermal repositories (specifically the SMU repository as well as the AASG repository) provided that these repositories comply with the standardized CSW and make it freely available.

Still, NGDS provides more capabilities than an aggregating CSW catalog. It also provides the future default solution for geothermal repositories and helps the DOE to build a grid of geothermal data repositories. The content of all repositories in this grid is searchable via a federated search mechanism that gives the user the possibility to execute faceted searches across all repositories, evaluate the detected datasets, and make them available for download1. The federation makes searches transparent to End Users/Data Consumers that have access to all data in the NGDS grid of repositories through any node of the system.

The NGDS End Users/Data Consumers are a variety of users that include legislators, federal and state agencies, financial investors, researchers, educators and students, interested public in general and industry representatives.

#### Goals and Context

The following are the basic goals of NGDS:

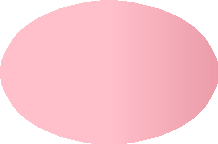
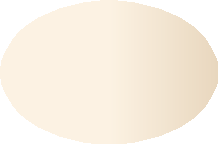
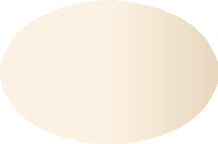
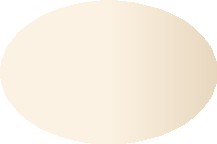
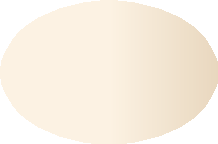
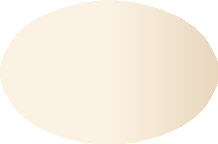
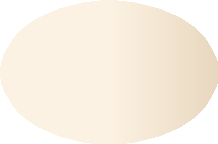
1. NGDS enables data providers the ability to create and administrate a repository of geothermal data.

1 Datasets are made available for download only if they are freely accessible.

1. NGDS enables End Users/Data Consumers to easily search geothermal data across a multiple of repositories of geothermal data.
2. NGDS enables End Users/Data Consumers to evaluate discovered data.
3. NGDS enables End Users/Data Consumers to select data to download.
4. NGDS in conjunction with other NGDS compatible tools enables End Users/Data Consumers to analyze downloaded data.
5. NGDS enables Software developers to extend NGDS functionality with new features.

Figure 3 depicts the basic goals of NGDS.

**uc Functionality**



**Data Ow ner Administrator**

**SW Dev eloper**

NGDS

**Maintain Repository**

**Maintain Central Node**

**Search across all repositories**

**Ev aluate Data Acquire Data Analyze Data**

**Add Features**

**Data Stew ard**

**Data Consumer**

*Figure 3: Basic Goals of NGDS*

###### Need for a Standard Data Repository

As outlined above, the activities lead to a variety of software systems for archiving geothermal data. However, the DOE requires a default software system to be used for future geothermal data collection projects. This software system is referred to as a “Node-in-a-Box”.

A Node-in-a-Box is a simple way for setting up a geothermal repository. The repository is simple to administrate, flexible with respect to configuration and adaptation by the data collector, and relies on standard technologies. Most importantly, it allows for federating its content as part of a grid of repositories for geothermal data.

Besides enabling data collectors to store their geothermal data, the NGDS provides a minimum set of housekeeping features for system monitoring, user management, logging of activities, support for backups, and basic security. The system has a number of basic features to adjust access rights to data within the system, distinguishing between End Users/Data Consumers (readers that can access and search for data

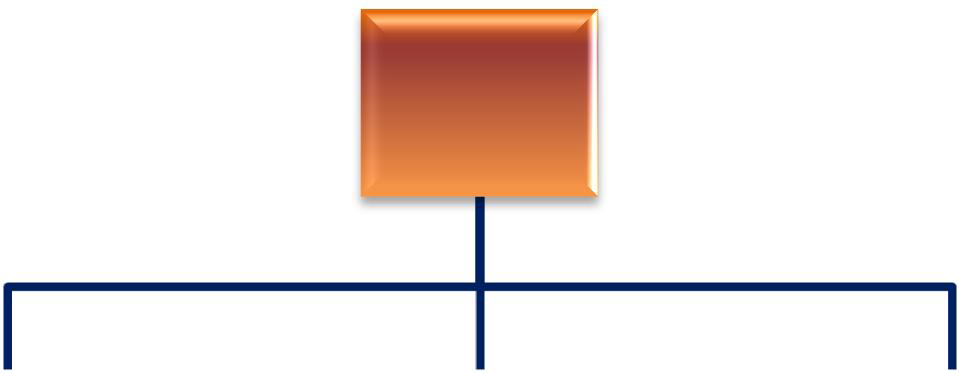
published in the system) and data providers (writers that can provide new content and modify existing data and metadata).

Also, the NGDS Node-in-a-Box supports basic business process for uploading, evaluating and publishing datasets. This process may involve multiple users, e.g. one user responsible for uploading data, and a second user responsible for reviewing the uploaded data and making it publicly available (or rejecting it).

###### Search across Multiple Repositories

The NGDS allows for searching through the central catalog of datasets within the grid of federated NGDS nodes (see Figure 4). This catalog, which indexes the data across all nodes, is accessible by each participating node.

**Central Catalog**



**Data Repository**

**Data Repository**

**Data Repository**

*Figure 4: NGDS is a grid of repositories*

End Users/Data Consumers utilize the NGDS user interface for executing faceted searches that combine many different search filters such as geographic region, type of data, data provider as well as keywords. The found results are visualized in an appropriate user interface screen. The most important aspect of the NGDS user interface is that it visualizes the found datasets in a map in an appropriate way (if found results can be geo-located and displayed appropriately). The map allows for the usual map features such as panning and zooming. Also, the user interface displays metadata for the found search results in an appropriate way and be able to sort the tables and to export them in an appropriate file format (typically CSVs).

###### Data Evaluation

Data analysts have the possibility to quickly review and evaluate found data sets. They may do this, for example, by relying on peer ratings of these datasets (ratings given by other users). Hence, data analysts are enabled to rate a dataset as well. They may also want to give ratings to the origin of data. Analysts are also able to triangulate regions of interest by overlaying information from multiple datasets, utilizing the geo-location associated to each dataset.

###### Data Acquisition

The metadata contains a URL for accessing the complete datasets. In case of unstructured data, the URL points to resources such as a PDF, TIF, JPEG or other type of file. In case of structured data, links for the supported services (WFS, WMS, and WCS) are provided allowing the complete dataset to be downloaded from the originating server. The NGDS user interface provides for downloading datasets in the appropriate form.

###### Data Analysis

Data Analysis is the strength of existing commercial and open source tools. The NGDS user interface provides some very basic features for analyzing data. However, it is not the goal to develop a full-fledged data analysis tool. Therefore, data analysis is limited to some very basic features such as display data in tables, and layering of different types of data as a way to improve data discovery. All other data analysis will be performed by third party tools that utilize the downloaded dataset for their particular use.

#### Significant Driving Requirements

The following table shows the architecturally significant functional requirements:

|  |  |  |
| --- | --- | --- |
| *Requirement* | *Description* | *Impact on Architecture* |
| Decentralized Repositories  CSW Support  Metadata Support | Support for independent repositories that store data. Support for a centralized catalog aggregating the catalogs of the distributed repositories.  Support for importing catalogs from any repository that exports its catalog via CSW (harvesting).  Support for all metadata fields that are collected about datasets and resources.  Support shall include harvesting of metadata and search of the metadata. | The NGDS SW architecture needs to be configurable to provide functionality for a Node-in-a-Box repository or a Central Node repository.  The framework chosen has support for CSW import/ export. CKAN supports import while export is supported via a third party extension.  The framework chosen has support for configuration of metadata. CKAN allows for changing the default metadata fields. |

Data Storage Support for storing resources

(files) of arbitrary type. The files are available for download.

The framework chosen has support for upload and download of arbitrary files and intelligent storage algorithm of files on the server. CKAN provides various algorithms out of the box.

OGC Support The content of structured data

files is available as WFS and WMS.

Integration of a solution for OGC services is required: The NGDS architecture uses Geoserver because it provides the reference implementation for OGC services.

Faceted Search The UI allows for using facets

to filter search results.

CKAN brings support for facets but it does not provide the configurability of facets out-of-the-box. NGDS implements this functionality as a UI extension.

Map Search The UI allows for using a map

to search for search data (geographic search).

CKAN does not support a map out- of-the box but there are various adaptations of CKAN that integrate a map.

For the architecture, this only impacts the UI. Leaflet is used as a map widget.

*Table 2-1: Architecturally significant functional features*

The following table outlines the most architecturally significant quality features:

|  |  |  |
| --- | --- | --- |
| *Requirement* | *Description* | *Impact on Architecture* |
| Sustainability | It is expected that the product will have a life span that is much longer than this project lasts. Therefore it is important to find a way how the product can be maintained after the project funding is used up. | The system must be easy to take over and maintain by an open source organization. The architecture promotes this by focusing on popular open source frameworks and libraries as a basis. The architecture also includes nose tests as a test framework. This is popular among |

python projects.

Implement ability

Funding and development time are both very limited. It is therefore required to find a way how the multitude of features can be realized with maximum development efficiency. The solution should make use of many existing frameworks in order to speed up development. The choice of these frameworks and platforms depend upon the team’s experience with platforms as well as feature completeness play an important role.

Since CKAN is the base platform that provides a high percentage of the functionality. Other functionality is integrated by combining other existing open source projects such as Geoserver rather than developing these functionalities from scratch.

Performance The system is able to

efficiently search through hundreds of thousands of datasets. Key performance bottleneck is the ability to execute a geographic search rapidly. Another factor is the ability of the system to harvest from nodes with reasonable execution speed.

NGDS is not expected to have many users in parallel (between 10 – 20 parallel users). Therefore scaling up to many users will not be an issue.

NGDS relies on PostGIS built-in capability for geographic search. This is most likely going to lead to bottlenecks. Therefore an algorithim might be integrate that uses a bucket mechanism to reduce the number of datasets that need to be searched. This mechanism might not be completed due to budget restrictions.

*Table 2-2: Architecturally significant quality features*

### Further Reading on Requirements and Functionality

The NGDS team has created sever al documents that provide an insight into the required functionality.

First of all, the requirements are documented in a requirements engineering document [REQ-2.7]. In addition the team maintains an Excel Spreadsheet that outlines the high- level Features of the system [FEATURES]. It groups the features into 7 feature groups that are listed in the following table.

|  |  |  |
| --- | --- | --- |
| *Feature Group* | *Description* | *# Features* |
| Contribute Datasets | This feature group contains all features related to | 6 |
|  | making datasets available via a NGDS Node-in- |  |
|  | a-Box. It covers uploading of different types of |  |
|  | resources either individually or in bulk mode |  |
|  | and making data available as OGC services. |  |
| User Administration | This covers the standard management features | 3 |
|  | for a Web application and specifically user |  |
|  | management. |  |
| Harvesting | This covers the require functionality to harvest | 2 |
|  | from arbitrary nodes using CSW. |  |
| Library Search | This covers various search features that do not | 5 |
|  | rely on the map. |  |
| Map Search | This covers the remaining search features that do | 10 |
|  | rely on the map. |  |
| Explore Dataset Details | This covers various features that allow to explore | 5 |
|  | the content of an individual dataset |  |
| Configuration | This covers the features required to adapt NGDS | 3 |
|  | to specific usage scenarios and updates of |  |
|  | standards. |  |
| General UI | This covers features that have to do with the | 3 |
| functionality of the UI. |
| **Total** | 37 | |

*Table 2-3: NGDS Feature Groups*

The development of the system is executed in an agile development process and this process is driven by use case stories. Sever al stories areaggregated into a so-called epic story. The development process and the epic stories are tracked in an Excel Spreadsheet [EPICS].

### Solution Background

This section describes the rational of the chosen architecture.

The solution space for the NGDS architecture is limited by two key quality requirements:

* + 1. Implementability
    2. Sustainability

Budget and time restrictions (Implementability) alone prohibit an implementation from scratch. Moreover, a very quick study of the field proves that there are various existing frameworks or solutions available that provide a large subset of the functionality. It was therefore clear from the beginning on that a green-field approach would not be efficient. The second factor (Sustainability) requires choosing a solution that has good chances to be picked up by open source communities. Of course, chances are higher if NGDS would piggyback on an existing and successful open source project that already has a large community.

Finally, the customer’s (DoE) preference for a certain framework (CKAN) had a high impact. The reason for the customer to prefer this framework is that it has been selected as the customer’s preferred platform for other projects besides from NGDS.

#### Architectural Approaches

As mentioned above the customer has a preference for the framework CKAN for strategic reasons. However in order to ensure that the strategic requirement does not collide with the specific requirements of NGDS, CKAN and various other framework candidates were verified against NGDS key functional and non-functional requirements. This was done in a two-step approach. A list of candidate solutions and frameworks was researched and determined. Afterwards the candidates were reviewed by a team of four stakeholders (Steve Richard, Ryan Clark, Paul Bruschi, and Christoph Kuhmuench). The results of the review were collected in an Excel Spreadsheet and proved that CKAN was indeed the best fitting framework.

#### Analysis Results

In the following candidate frameworks and their rating with respect to functional completeness and non-functional rating are listed:

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Framework* | *Functional Completeness* | *Non Functional Rating* |
| 1 | CKAN | 80% | 75% |
| 2 | GTDA | 76% | 68% |
| 3 | AASG | 72% | 55% |
| 4 | Geonode | 54% | Not rated |
| 5 | DSpace | 62% | Not rated |
| 6 | DataONE | <50% | Not rated |
| 7 | Cuashi | <50% | Not rated |
| 8 | SISSweb | <50% | Not rated |

*Table 2-4: Rating of Framework Candidates*

Although the two frameworks GTDA and AASG have a high rating, CKAN was chosen because it was more complete regarding features, and is the framework preferred by the customer. Note that the frameworks do cover large percentage of the functionality required for NGDS which is the main reason to base further development on one of those frameworks.

### 2.4 Product Line Reuse Considerations

NGDS is tailored towards managing arbitrary unstructured and structured data. NGDS does provide some functionality specific to the geothermal domain. However, NGDS is designed in such a way that it could be tailored for other domains that require storage and access of distributed data.

## 3 Use Case Scenar ios

This section contains sever al use case scenarios that are used to show the interaction between the components of the system. Use case scenarios are a good way to demonstration how requirements are fulfilled by the chosen architecture. Use case scenarios further help developers and architects to understand the dynamics of the system.

Note that this document employs use case scenarios and not use cases. In order to prevent confusion between *use cases* and *use case scenarios* our definition of both follow:

###### Definition Use Case:

*A use case represents the actions that are required to enable or abandon a goal. A use case has multiple “paths” that can be taken by any user at any one time.*

###### Definition Use Case Scenario:

*A use case scenario is a single path through the use case.*

The seven most important use cases scenarios (simply called scenarios for the remainder of this section) are described. The chosen scenarios touch many of the NGDS requirements and reflect the most frequently used functionality and the most important ones from the customer’s perspective.

The scenarios cover all seven main functionality fields [FEATURES] of NGDS. The scenarios depict successful paths of a use case (e.g. only successful upload creation of a dataset) because the non-successful path is usually very simple because it only requires discarding any uploaded information.

|  |  |
| --- | --- |
| *#* | *Scenario Description* |
| 1  2  3  4  5  6 | Successful Upload of In this story the user creates a new dataset, enters all unstructured Dataset relevant metadata, adds one resource with unstructured Resource content to the dataset, adds the remaining required  metadata and publishes it.  This story assumes that the user has the role Data Steward and thus has the right to create and publish the data.  Successful Upload In this story the user creates a new dataset, enters all Structured Dataset relevant metadata, adds one resource with structured Resource content that follows a certain content model to the dataset,  adds the remaining required metadata and publishes it.  This story assumes that the resource validates and that the user has the role Data Steward and thus has the right to create and publish the data.  Successful Publish In this story the user selects a dataset that has at least one structured Resource structured resource. The user publishes one of the  as OGC Services structured resources as OGC services by clicking on the  “Publish as OGC” Button.  This story assumes that no errors occur while publishing.  Map Search In this story the user marks an area on the map, enters some search keywords and clicks on the search button (or presses the “Return” key).  This story assumes that datasets matching the query are found. They are displayed on the map and in the results list.  Keyword Library In this story the user enters some keywords into the search Search field and presses the search button (or presses the “Return”  key). This story assumes that sever al (more than one) results are found. The user then activates a facet in the facet UI Widget to filter the result.  Successful Upload In this story the user creates a new dataset, enters all Shapefile relevant metadata, adds one shapefile as a resource that  follows a certain content model to the dataset, adds the remaining required metadata and publishes it. |

This story assumes that that the shapefile validates and that the user has the role Data Steward and thus has the right to create and publish the data.

7 Successful Harvest Metadata

This story assumes that the user has already used the UI of the Central Node to schedule harvesting from a certain node. Therefore in this story the Linux CRON damon represents the “user”. Upon the scheduled time the cron damon triggers that the Central Node contacts the node and requests an update of its content catalog using CSW and integrates the returned catalog update into its own catalog.

*Table 3-1: Use Case Scenarios*

## M ain Decisions

### One Software for both Node-in-a-Box and Central Node

NGDS developed only one software stack that provides the functionality for both central node and Node-in-a-Box. The reason for this decision is that so many features of both node types are the same that it is easier to develop one software stack and configure the behavior of the system with a configuration file. In order to do so a new parameter to CKAN’s configuration file (“development.ini”) that defines the behavior of a node during startup was added. The node can either be configured as Central Node or Node- in-a-Box. As a Central Node the NGDS software provides the harvesting but no uploading capability while as a Node-in-a-Box the NGDS software provides content uploading but no harvesting capability.

### Operating System Support

Ubuntu 12.04 LTS was chosen as the NGDS reference platform. Due to the nature of NGDS, most users will most likely want to run it on a Linux Operating System. Ubuntu is a well-known and well-documented Linux OS. Also, CKAN is optimized for Ubuntu or other Debian-based Linux distributions.

NGDS has been developed with Ubuntu Version 12.04 LTS because it has a long support cycle. When the next LTS version becomes available the NGDS development environment will be upgraded. Installation files, etc. are written in such a way that they will likely work on any Debian-based Linux version. Therefore, porting to other Linux platforms is a minor effort. NGDS is also frequently tested and installed on the Mac OS X platform but NGDS is not test installed the system on Windows. Most likely it is possible to start the system on Windows but to fully support that would drain too many development resources and it is questionable if this form of testing adds much value.

### Base Backend Technologies

NGDS relies heavily on Python and CKAN. This defines the architecture of NGDS to a great extend because CKAN is a framework with well-defined extension points where NGDS functionality has been added. Further, CKAN is used as-is without modifying the CKAN core. Bugs are found in the CKAN core are reported and tracked through the Open Knowledge Foundation’s support team and their tools.

Since CKAN is targeted towards Postgres, NGDS’s reference database is also Postgres. The PostGIS extension of Postgres is used for geographic features.

In order to serve OGC services NGDS uses Geoserver which runs in its default setting on jetty.

For indexing of Metadata and full-text indexing NGDS uses SOLR (on jetty). SOLR is configured according to the CKAN recommendations.

### Base Frontend Technologies

NGDS uses HTML5 and CSS3 as the base technology for the frontend. HTML5 and CSS3 are now supported by all major browsers (even later versions of Internet Explorer). In addition NGDS uses various JavaScript Libraries. Specifically, NGDS uses JQuery and various libraries based on JQuery. Furthermore, NGDS uses CSSless in order to reduce CSS complexity. At the time of writing of this document these are the state-of-the-art technologies for developing Web applications.

For the production of the HTML pages, NGDS uses the Jinja2 templating system that is built into CKAN. The templating system can be compared to PHP and allows to bring backend-information (made available via Python) into the frontend HTML content.

Jinja2 is very flexible and works well in the CKAN environment. The reason for using Jinja2 is that it is the reference solution for CKAN. Building other templating systems into CKAN would be extra effort and causes problems regarding maintainability.

In many cases it is possible to create the page by either using Jinja2 templating or JavaScript. Jinja2 is preferred rather than JavaScript whenever possible because Python code and Jinja2 code is in general easier to maintain than JavaScript code.

The Leaflet Map Widget is used to present maps ([http://leafletjs.com/).](http://leafletjs.com/)) This widget is currently popular and has an active developer community. OpenLayers (<http://openlayers.org/)> was considered but due to higher experience with Leaflet among the NGDS developers, it was decided to go with Leaflet.

Naturally it is not possible to simply replace Leaflet with another map widget solution. However to the extent possible, the Leaflet-specific code is encapsulated in separate classes and modules so that the classes can be replaced in the future in case that the map widget needs to be exchanged.

## High-level Context Views

This section contains the core technical information about how NGDS is designed and developed. As described in the document preface, the Kruchten’s 4+1 View Model and added a set of System Context Views are used. The system context view depicts the external components that NGDS has to cooperate with and the protocols used.

### System Context Views

NGDS is a grid of data repositories containing datasets of geothermal information. Each of these repositories maintains a catalog listing all of its (publicly) available datasets.

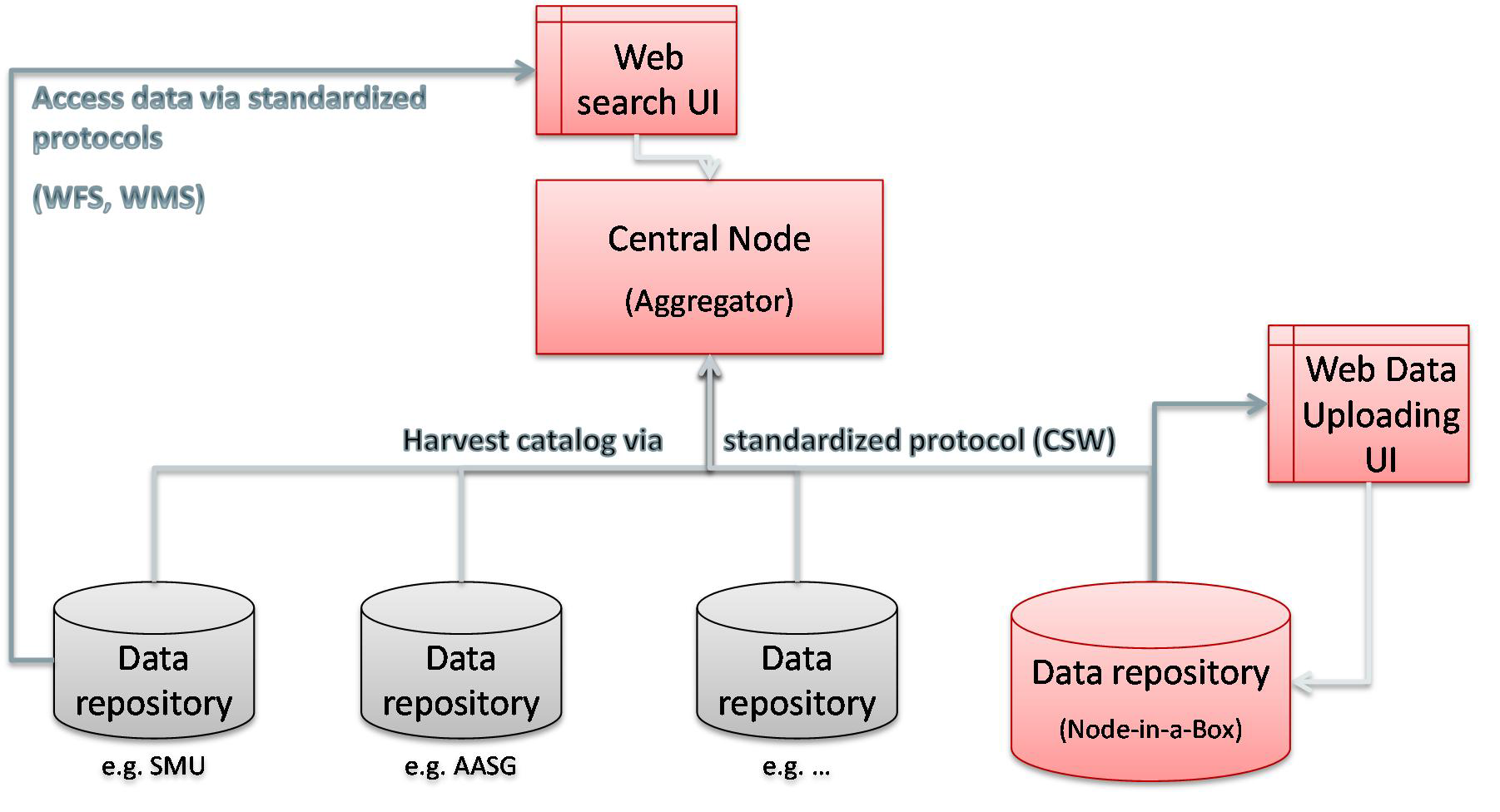
This catalog must be made available via the standardized protocol “Catalog Service for the Web – CSW”. The catalog must provide metadata information about each dataset according to the ISO metadata standard ISO 19115.

NGDS provides a reference implementation for such a data repository (the so called “Node in a Box”) supporting CSW. However, institutes can rely on other solutions as long as they provide the catalog using CSW and according to ISO 19115.

There is at least one node in the system playing the role of the central access point that aggregates the catalogs of the data repositories. Note that it only aggregates the metadata provided in the catalog. It does not duplicate the datasets. These remain on the repositories.

The Central Node also provides a Web-based UI that allows user to search, preview, and download data. The Web-based UI can also access structured data via standardized protocols WFS and WMS and must provide viewers for these protocols.

Figure 5 depicts the interaction between Central Node and repositories.



*Figure 5: NGDS System Context View*

### Logical Views

In the NGDS documentation the logical view package contains various view angles. A domain model represents an abstract view of the NGDS domain.

The solution domain shows the high-level building blocks of the architecture. First, a static view of the high-level building blocks and then a dynamic view describe sever al interactions between the components according to user scenarios.

#### Domain Model

The Domain Model of NGDS can be represented as a class diagram (see Figure 6).

**class Class Diagram View**



**Anonymous**

**OGC Protocol**

i mplem ents

**Data Consumer**

**Node-in-a-Box**

\*

**Dataset**

\*

**NGDS User**

**Node**

\* \*

harvest

**Data Submitter**

\* \*

**Aggregator**

**Resource**

**Data Ste w ard**

**Interaction**

**Administrator**

**Upload Publis h Search**

**Social Netw orking**

**Administrate**

*Figure 6: NGDS Domain Model as a Class Diagram*

NGDS distinguishes between five user roles interacting with the system. In a similar fashion the five types of interaction between the user and the system are represented by different nodes.

Anonymous users will have limited access to NGDS use the system to search for datasets. An anonymous user has no account and therefore cannot log into NGDS and cannot be identified.

End Users/Data Consumers will make use of the search interactions and the social networking features. A Data Consumer has logged into the system and can therefore be recognized. This is required in order to allow access to the social networking features. Note that any user can sign up to NGDS and gain the privileges of this role.

Data Submitters will make use of the search, social networking, and upload interactions. A Data Submitter has the extra privilege to upload data. Note that uploaded data stays invisible to NGDS End Users/Data Consumers until a Data Steward makes them “public”. This process shall give Data Stewards the opportunity to ensure the quality of the system.

Data Stewards will make use of the search, social networking, uploading, and publishing interactions. As stated above the Data Steward is responsible for data quality.

Administrators will make use of all interactions including the system administration features.

As already outlined, the system distinguishes between two node types: repository nodes

– also called “Node-in-a-Box” – store datasets and aggregating nodes – also called “Central Node” – allow for searching across all repositories.

A dataset represents a unit of geothermal information. The core of this information is stored in the resources contained in this dataset. In addition a dataset maintains metadata about this core information (e.g. location on the map, author, title of the resource, an abstract of the contained data, etc.). Essentially, this metadata shall help users to identify datasets that are interesting for their problem domain.

NGDS supports various types of resources as outlined in the following Figure 7.

**class Class Diagram View Resources**



**Resource**

**Offline Resource Online Resource**

**Structured Resource**

**Unstructured Resource**

**Non-Content Model Resource**

**Content Model Resource**

*Figure 7: NGDS Domain Model of Resources as a Class Diagram*

Offline Resources are entries that are non-virtual. Examples are cores, non-scanned maps and other tangible artifacts. Online resources can be in general downloaded as files or accessed as services. Structured and unstructured resources are distingushed between each other. Further, the structured resources are split into non-content model

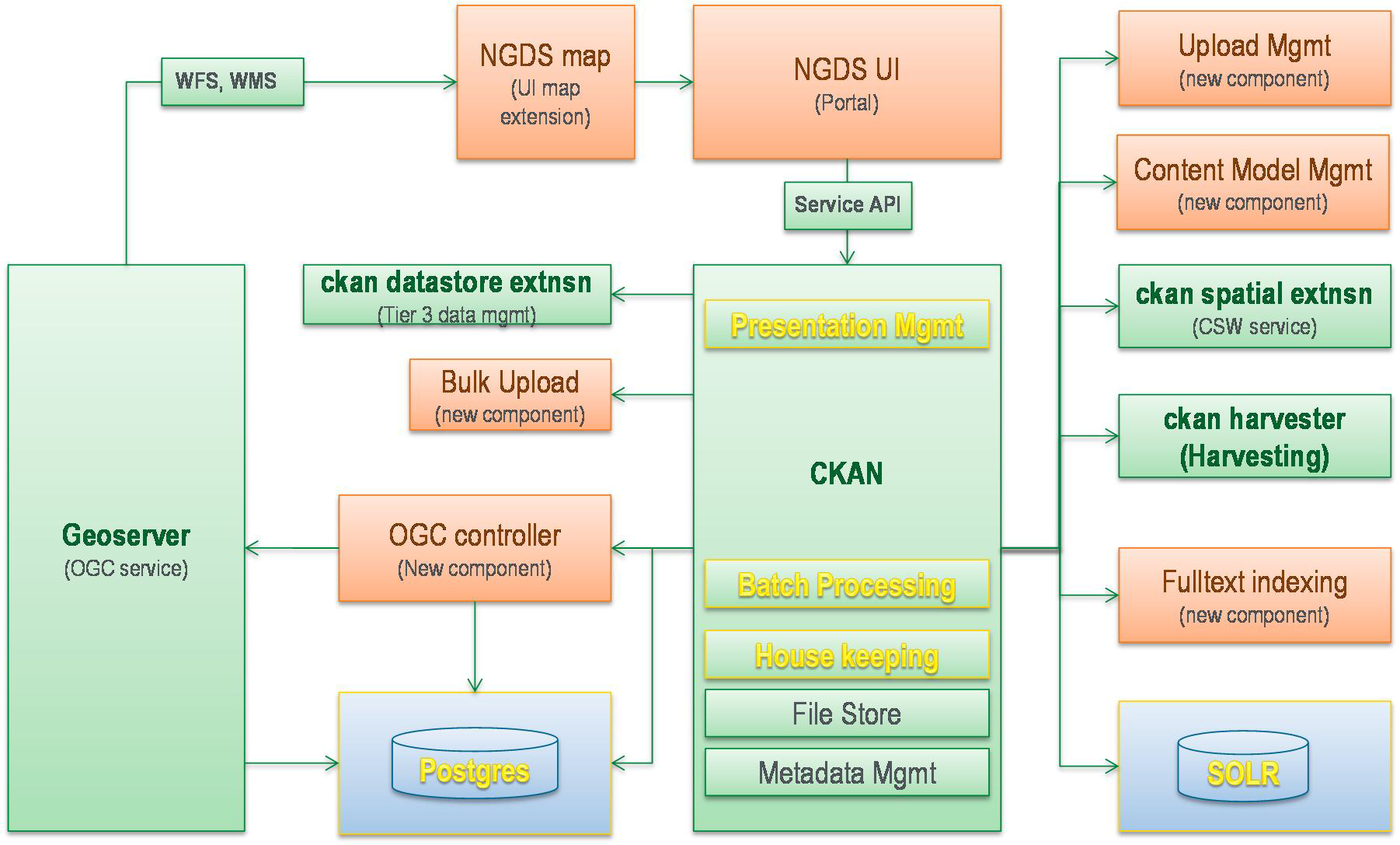
resources and content model resources. The latter ones follow a well-defined template and NGDS can validate the resource against that content model.

#### High-level Components

As discussed before NGDS is based on CKAN and therefore the system architecture is largely defined by this framework plus additional tools and components. Figure 8 outlines the high-level components. Note that not all used CKAN extensions are shown in this figure. A complete list of all extensions can be found in section 5.3.1 on page 34.

Note that components that are colored in green are existing components that may be configured for NGDS purposes. The green-colored components were not modified by the NGDS development team other than modifying configuration files to adapt them. Components that are colored in orange are new CKAN extension components developed by the NGDS team for NGDS. All components developed by the NGDS development team are realized as CKAN extensions.

Components colored in blue are base software that is not modified for NGDS.



*Figure 8: NGDS High-level Components*

The following sections discuss the components in more detail.

##### CKAN

CKAN is the core framework of NGDS and provides a large percentage of the functionality. CKAN is a Python based data management system that provides a plug-in mechanism allowing for extending and tailoring it to specific needs. Besides from the

core features File Storage and Metadata Management it also provides a multitude of features that are only indirectly features. Examples for those components are batch processing, housekeeping, Presentation Management and Security.

##### SOLR

SOLR is an indexing framework developed and maintained by Apache. Essentially it can index data and provides a search API to find data that contains that keyword. SOLR is a crucial element of CKAN. CKAN uses SOLR to index the metadata of each dataset.

Essentially all search requests are forwarded from CKAN to SOLR which returns a list of the unique IDs of all datasets that have matching metadata.

SOLR is also used to do full-text indexing of uploaded text files.

##### Geoserver

Geoserver is the OGC’s reference implementation for the standardized protocols WCS, WFS, and WMS. It is based on Java and requires a servlet container to run. NGDS is currently based on Jetty.

Geoserver essentially accesses a SQL database, and can represent arbitrary tables in this database as WCS, WFS, and WMS, provided that they provide geo-information. NGDS uses Postgres/ PostGIS as relational database with geo-coding support. Geoserver provides a UI to control which tables are made available as a service but it also provides a RESTful API. NGDS uses this RESTful API to control Geoserver from a CKAN plug-in.

##### Postgres/ PostGIS

Postgres plays the role of the relational database server. PostGIS is a Postgres extension that provides support for geographic features. Postgres/ PostGIS could be replaced with another relational database system provided it supports geographic features. But since Postgres/ PostGIS is the reference database system for CKAN, Postgres/ PostGIS is the reference solution for NGDS.

Postgres is used to store all Metadata. It also hosts dynamically created tables that represent the content of structured data resources (i.e. CSV files, XLS files, XML files). Essentially each column in the structured files is translated into a column in the dynamically created table.

The Postgres database also provides the input for Geoserver which computes the OGC services (WFS, WMS, and WCS) from the dynamically created tables that represent the content of the structured files.

##### CKAN Data-store and Data-storer Extension

The CKAN Data-store extension is an already existing extension of CKAN developed by the Open Knowledge Foundation. In short, the Data-store is the place to store structured

data. It is paired with a second extension called CKAN Data-storer Extension. The Data- storer is capable of parsing arbitrary XLS, CSV, and XML files and stores their content in the Data-store. In essence it creates a new table for each table in the uploaded structured file. The table is named after the unique id of the file.

The CKAN Data-store provides a RESTful API to access the data in those dynamically created tables. Various CKAN UI extensions provide previews of the data, e.g. as searchable and sortable tables (in HTML), Graphs and Charts, or as an overlay on a map. These CKAN UI extensions make use of the API provided by the Data-store.

##### OGC Controller Extension

This component is developed by the NGDS team. It is realized as a CKAN extension. It is responsible for connecting the CKAN data-store with the Geoserver and thus allows a Node-in-a-Box to export structured resources as OGC services.

##### Upload Management

This component is developed by the NGDS team. It is realized as a CKAN extension. It is responsible for managing the additional metadata fields that NGDS requires on top of the metadata already managed by NGDS. It also provides a UI that guides the user through the process of uploading one individual file.

##### Bulk Upload Management

This component is developed by the NGDS team. It is realized as a CKAN extension. It is responsible for managing the upload datasets in a bulk format, i.e. it allows a user to upload multiple datasets in one step. In principle the user has to generate an Excel file containing the metadata for each resource that shall be uploaded. The all resources to be uploaded are stored in a ZIP file. A simple UI allows the user to provide both Excel and ZIP file. The system processes the uploaded content in a batch process and provides the user with information about the progress and if required error situations (e.g. incomplete metadata).

##### Content Model Management

This component is developed by the NGDS team. It is realized as a CKAN extension. NGDS supports geothermal-specific content models. Essentially a content model provides a template for certain data resources (in general this is an Excel file).

Content Model Management verifies the data of a resource. If the uploaded resource does not comply with the content model it will reject it and provide the user with an error message.

##### CKAN-Spatial Extension

The CKAN-Spatial extension is an already existing extension of CKAN developed by the Open Knowledge Foundation. The NGDS team modified the configuration of the extension so that it supports ISO metadata.

In general this extension allows for exporting the content of the NGDS metadata as a CSW service. This functionality is only required on a Node-in-a-Box.

##### CKAN Harvesting Extension

The CKAN-Harvesting extension is an already existing extension of CKAN developed by the Open Knowledge Foundation. The NGDS team modified the configuration of the extension so that it supports ISO metadata.

In general this extension allows for reading metadata catalogs that are either provided in the CKAN native protocol format or via CSW.

This functionality is only required on a Central Node.

##### CKAN Full-text Indexing Extension

This component is developed by the NGDS team. It is realized as a CKAN extension. The component provides full-text indexing functionality. It is hooked into the CKAN file upload mechanism. When a file is uploaded that is of one of the supported file formats (at the moment only text and PDF) the full-text indexing component will create a full- text index of the file content using SOLR. The content is linked the ID of the dataset so that search results will include hits into the full-text index.

##### NGDS UI

The NGDS UI is also hooked into the system as a CKAN extension. It makes use of the templating system (Jinja2) to create a UI according to the specifications of the NGDS design team (Siemens UX team and Anthro-Tec). The NGDS UI simply overrides the CKAN default UI by using CKAN’s UI override mechanisms. In general overriding makes use of two mechanisms: First of all it overrides the NGDS default look & feel by overriding the CKAN Cascading Style Sheets and header and footer templates. In addition it also overrides the main content of those pages for which the design team has created a Look and Feel that is significantly different from the CKAN default look & feel.

##### CKAN Map Extension

The CKAN Map Extension is part of the NGDS UI component. It is also hooked into the system as a Jinja2 template. However, due to its dynamic nature it is heavily JavaScript oriented. The corner piece of the component is the Leaflet Map Widget ([http://leafletjs.com/).](http://leafletjs.com/)) It is a very intensively developed widget and provides many

features. It provides the capability to display map base layers and display WMS maps on top.

#### Layered View

A layered view provides next level of detail insight to NGDS. Note that a layered view is still a high level of abstraction that may lead to misunderstandings. Figure 9 shows NGDS as a layered block diagram. If a block is placed on top of another block this means that the upper block calls the API of the lower block. Neighboring blocks may use each- others API to communicate. There should not be any communication “upwards”, with the exception of callback functions and methods as this would create circular dependencies.



*Figure 9: NGDS Layered View*



The following components are added to the component diagram shown in the previous subsection:

##### Pylons

Pylons is the base framework of CKAN. It provides a multitude of functionality for Web Applications. Details about the framework can be found here: [<http://www.pylonsproject.org/>]

##### SqlAlchemy

SQLAlchemy is CKAN’s Object Relational Mapping Framework (ORM). It is a framework that is used in many Python projects. Details about SQLAlchemy can be found here: [<http://www.sqlalchemy.org/>]

##### Jinja2

Jinja2 is the templating system used by CKAN. It provides features that are similar to PHP and it is well-integrated with Python, i.e. it allows the transfer of data between the Python environment and the templating system. Details about Jinja2 can be found here: [<http://jinja.pocoo.org/>]

##### Virtualenv

Virtualenv allows creating a virtual Python environment. With virtualenv one can create a Python Base Directory in Linux. All files required to run the Python interpreter are copied from the default directories of the operating system into that folder. When additional libraries are installed from the internet (e.g. via PIP) they are installed into that directory and not in the default directory of the operating system. Details can be found here:

[https://pypi.python.org/pypi/virtualenv]

##### PIP

PIP is a package management system used to install and manage software packages written in Python. It is the default tool for managing Python libraries. Details can be found here:

[https://pypi.python.org/pypi/pip/]

##### CKAN, CKAN RESTful API, and CKAN Extension Interface

The components CKAN Core, CKAN RESTful API, and CKAN Extension Interface comprise the CKAN system. The CKAN RESTful API and the Jinja2 templating System can be used to develop a new API on top of CKAN. The CKAN Extension Interface on the other hand provides the means to develop extensions in Python that extend the functionality of CKAN. It also allows for hooking into the existing CKAN API, e.g. by adding additional code into the CKAN file upload mechanism and other core features.

Details can be found here: [[http://CKAN.org](http://CKAN.org/)]

##### CKAN Extensions

The CKAN extensions make use of the three mechanisms CKAN RESTful API, CKAN Extension Interface, and Jinja2, in order to provide additional functionality. There are already many extensions available that are maintained by either the OKFN or other organizations. Some extensions are installed with the CKAN core package; others need to be installed separately. NGDS makes use of the following CKAN extensions:

|  |  |  |
| --- | --- | --- |
| *Extension* | *Description* | *URL* |
| CKAN-ext-data- store (data-store)  CKAN-ext-data- storer (data- storer)  Statistics (stats)  Data Viewer (json\_preview, resource\_proxy, recline\_preview)  Geospatial Capabilities (spatial\_metadata  , spatial\_query) | The data-store allows for storing and accessing structured data via a RESTful API. It is part of the CKAN installation package  The data-storer stores structured data in the data-store. It reads structured files (CSV, XLS) and stores them in the data-store.  CKAN’s stats extension analyzes your CKAN database and displays sever al tables and graphs with statistics about your site, including:   * Total number of datasets * Dataset revisions per week * Top-rated datasets * Most-edited Datasets * Largest groups * Top tags * Users owning most datasets   CKAN provides an API to create previewers for data resources. It also provides a number of plug-ins for structured data. PDF, text and images are simply presented within an iFrame. For structured data it provides a previewer based on recline.js. The  plug-ins recline\_preview, resource\_preview, json\_preview, and resource\_proxy are part of the Data Viewer.  CKAN offers a set of geospatial features that allow adding spatial information to your datasets.   * Spatial\_metadata: Associate geographic data with datasets, | [http://CKAN.org](http://CKAN.org/)  https://github.com/okfn/C KANext-data-storer  <http://docs.CKAN.org/en/> latest/stats.html  <http://docs.CKAN.org/en/> 769-docs-reorg/data- viewer.html  <http://docs.CKAN.org/en/> CKAN-  1.7.2/geospatial.html |

with automatic geo-indexing of datasets.

* + - * + spatial\_query: Search for datasets using spatial queries.
        + Wms\_preview:

CKANext-spatial (wms\_preview)

This extension contains plug-ins that add geospatial capabilities to CKAN. It contains a large number of plug-ins of which only a small subset are used.

* Wms\_preview: provides a previewer for WMS resources.

https://github.com/okfn/C KANext-spatial

*Table 5-1: Existing CKAN Extensions used in NGDS*

The following table lists the extensions developed by the NGDS team. The last column maps the extension to the component described in the logical view. In general one component from the logical view in Figure 8 comprises of one or more CKAN extensions.

|  |  |  |
| --- | --- | --- |
| *Extension* | *Description* | *Contributes to Components* |
| OGC-  controller (Geoserver)  NGDS  Metadata (metadata)  NGDS  Harvesting (ngdsharvest)  ngdsui | Manages the connection to Geoserver and makes WMS, WFS, and WCS services available.  Handles the NGDS-specific Metadata  Handles the harvesting.  Serves the UI. Currently also includes other features (planned to be refactored) | OGC-controller  Upload Mgmt  CKAN Harvester  Ngds map, ngds ui, upload mgmt., content model mgmt., full-text indexing |

*Table 5-2: New CKAN Extensions developed by the NGDS team*

### Development Views

#### Component View of CKAN

This section describes some of features and components of CKAN. The API of the system is well-documented and can be found here: [http://CKAN.org.](http://CKAN.org/)

CKAN is based on a multitude of Python frameworks and libraries. The glue between all components is provided by Pylons (<http://www.pylonsproject.org/)> which by itself again consists of a multitude of frameworks and libraries. Essentially Pylons provides the base functionality to create a Web application by developing extension, hooked into the core system. It makes use of SQL Alchemist for Object-Relational Mapping (ORM). It also provides all base functionality to create a Web Application (Session Management, Security, Forms Management, Templating Language, etc.).

Figure 10 outlines the most important Pylons libraries. Please refer to [PYLONS] for more details about the structure and components of Pylons.

**Beaker** (Caching, Session)

**Pylons**

(glues all together)

**WebError**

(Web Debugging)

**Decorator**

(@validate,

@jsonify)

**Routes**

(URL mapper)

**WebHelpers** (create HTML, and more web functionality)

**FormEncode** (validate web forms)

**Setuptools**

(easy\_install)

**WebOb** (pylons.request, pylons.response)

**Jinja2** (template language)

**Simplejson**

(json converter)

**SqlAlchemy**

(ORM)

**Nose**

(unit testing)

**AuthKit**

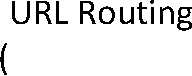
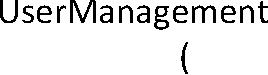
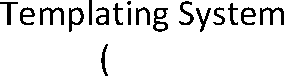
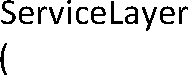
(Authentication)

|  |  |  |
| --- | --- | --- |
| **Paste, PasteDeploy, PasteScript** (Command Line  Tool) |  | **Tempita** (paste dependency, template  language) |
|  |

*Figure 10: Pylons Base Libraries*

Figure 11 outlines the most important components of CKAN.

*Figure 11: CKAN Component View*



Please refer to the CKAN documentation [[http://www.CKAN.org](http://www.CKAN.org/)] for more information about CKAN.

#### Github Code Overview

NGDS maintains it software in github.com. In the following the structure of the NGDS github presence is outlined:

|  |  |
| --- | --- |
| *Repository* | *Description* |
| ckanext-ngds  dev-info  documents  ckan-dev  ckanext-spatial- dev  ckan  ckanext-spatial | Most important of all NGDS repositories on github. Contains the source code for the NGDS extensions.  Contains information for developer. Also contains the installation scripts.  Contains development documents such as sprint plans, requirements documents, and design documents.  Copy of the CKAN repository. It is intended to capture any NGDS modifications of CKAN. Currently, there are no modifications.  Cop of the CKANext-spatial repository. Is intended to capture any NGDS modifications to this plug-in. Currently, there are no modifications.  Contains a fork of the CKAN project.  Contains a fork of the CKAN spatial extension project. |

*Table 5-3: NGDS Repositories on* [*http://github.com*](http://github.com/)

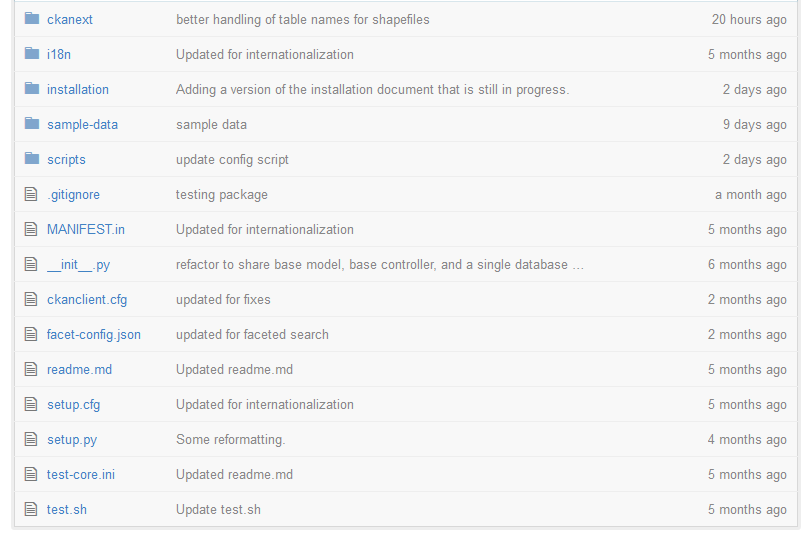
The NGDS source code can be found in two of the repositories mentioned above:

1. *ckanext-ngds:* This repository contains all code written by the NGDS development team. All code is written in a single CKAN extension that provides various plug- ins (see Table 5-2).
2. *ckan-dev:* This repository contains a copy of the CKAN source code. It is designed to also hold any modifications that NGDS team needs to apply to CKAN. However, up to now there are no modifications and the repository contains the code of CKAN version 2.0.
3. *ckanext-spatial:* This repository contains a copy of the CKAN spatial extension source code. It is designed to also hold any modifications that NGDS team needs to apply to CKAN. However, up to now there are no modifications.

The most interesting of the repositories to understand is the one in ckanext-ngds. The screenshot shown in Figure 12 outlines the structure of the repository.

The repository follows the standard approach for CKAN extensions. The majority of the code (i.e. the ngds plug-ins can be found in the directory CKANext. The directory i18n contains code and data for internationalization. The installation directory simply contains the word file that describes the installation procedure for NGDS. In the

directory sample-data a number of geothermal resources can be found that can be used to test NGDS content model functionality. The scripts directory contains a number of configuration scripts. Among the files in the directory setup.py shall be highlighted. It follows the standard approach for CKAN extensions and provides information about all plug-ins provided with this CKAN extension. The two test files are used to execute tests on the command line.



*Figure 12: Content of the base directory of the repository ckanext-ngds*

### System Dynamics Views

This section describes the major NGDS use cases and their dynamic aspects. The description uses message sequence charts to show which components are involved and how they interact.

#### Use Case: “Upload unstructured Dataset Resource”

In this use case the user creates a new dataset and adds an unstructured resource to this dataset. The task of the NGDS is to store the resource and associate it with the metadata. It also needs to update the search index so that the dataset can be found using the search mechanisms (Map Search, Keyword Search, and Faceted Search). Figure 13 shows the interaction between the components.

**sd Uploa d Unstructured Dataset Resource**

|  |  |
| --- | --- |
| CKAN Server | |
|  |  |

|  |  |
| --- | --- |
| Postgres | |
|  |  |

NGDS UI

SOLR



Data Submitter

CreateDataset(metadata\_name, metadata\_value)

CreateDataset(metadata)

CreateUpdateMetaData(metadata)

CreateUpdateIndexingData(dataset\_id, metadata)

RenderPage(PageName)

CreateResource(metadata\_name, metadata\_value)

CreateResource(datset\_id,

metadata)

CreateUpdateMetaData(metadata)

CreateUpdateIndexingData(dataset\_id, metadata)

RenderPage(PageName)

UploadOnlineResourceFile(file\_name)

UploadDataFile(DataFileType, DataFile)

StoreFile()

RenderPage(PageName)

*Figure 13: MSC for Use Case "Upload Dataset with Unstructured Resource"*

The user enters the relevant metadata into the CKAN UI which is then transferred to the CKAN core system. The CKAN core creates a unique ID for the dataset and stores the metadata information in the Postgres SQL database. Then it forwards the Metadata to SOLR for indexing. SOLR creates an index and assigns it to the dataset ID. When the user uploads the unstructured dataset it will be stored in the Linux file system. Note that

the metadata for the resource is also stored in Postgres and indexed by SOLR. This step is omitted in the figure above to simplify visualization.

#### Use Case: “Upload Structured Dataset Resource”

In this use case the user creates a new dataset and adds a structured resource to this dataset. The task of the NGDS is to store the resource and associate it with the metadata. It also needs to update the search index so that the dataset can be found using the search mechanisms (Map Search, Keyword Search, and Faceted Search). Furthermore it needs to verify the resource against its content model template and store the data in the CKAN data-store for retrieval and statistical analysis.

Figure 14 shows the interaction between the components.

**sd Upload Structured Dataset**

Data Submitter

CKAN Server Postgres SOLR

|  |  |
| --- | --- |
| NGDS UI | |
|  |  |

CKAN Data Store Extension

CKAN Data Store

Content Model Verification



CreateDataset(metadata\_name, metadata\_value)

CreateDataset(metadata)

CreateUpdateMetaData(metadata)

CreateUpdateIndexingData(dataset\_id, metadata)

RenderPage(PageName)

CreateResource(metadata\_name, metadata\_value)

CreateResource(datset\_id, metadata)

CreateUpdateMetaData(metadata)

CreateUpdateIndexingData(dataset\_id, metadata)

RenderPage(PageName)

UploadOnlineResourceFile(file\_name)

UploadDataFile(DataFileType, DataFile)

StoreFile()

VerifyFile(file\_name)

UpdateDatstore(file\_name)

ParseFile(file\_name)

StoreData()

StoreDatainNewTable()

RenderPage(PageName)

*Figure 14: MSC for Use Case "Upload Dataset with Structured Resource"*

In many aspects the use case causes the same interactions as uploading unstructured resources. The difference here is that Content Model Verification is involved to check the consistency of the uploaded file. Furthermore the CKAN data-store is involved to create a new table containing the structured data. The reason for adding the data to the CKAN data-store is that it can be statistically analyzed and also previewed using CKAN’s built-

in previewers for structured data. Also, uploading the data to the CKAN data-store is a prerequisite to make the resource available as OGC services.

#### Use Case: “Publish Structured Resource as OGC Services”

In this use case the user wishes to make a resource of a dataset available as OGC services (i.e. as WFS, WMS, and WCS services). Since it is a structured resource it is stored in the CKAN data-store, i.e. a table containing the content of the resource exists. It is the task of the NGDS software to add a new column of type PostGIS/GeoShape to that table and calculate its value for each row. It then needs to inform Geoserver about this table so that Geoserver starts serving it as WFS, WMS, and WCS.

Figure 15 shows the interaction between the components.

**sd Publish OGC Layer**



End User/ Data Consumer

NGDS UI CKAN Server Postgres Geoserver OGC

|  |  |
| --- | --- |
| CKAN Data Store | |
|  |  |

Controller

Spatialize Resource()

Spacialize(resource)

Spcialize(resource)

AddColumnForSpacializedInformation()

UpdateDBTable()

CalculateGeoValues()

StoreData()

StoreDatainNewTable()

CreateOGCServices()

FindMetadata(SearchCriteria)

*Figure 15: MSC for Use Case " Publish structured resource as OGC Services"*

The user triggers the process by pressing the “Spatialize” button in the NGDS UI on the dataset details page. The UI calls a RESTful interface which CKAN routes to the OGC Controller Plug-in. The plug-in then makes use of the CKAN data-store API which in turn adds the new column to the table associated with the resource. Then the OGC controller plug-in calculates the values for the new column and updates the table again indirectly by making use of the CKAN data-store API.

In the next step the OGC Controller plug-in calls Geoserver’s RESTul API. Geoserver then publishes the relevant OGC services using the table associated with the resource. The newly created column is used for geo-referencing.

#### Use Case: “Map Search”

In this use case the user wishes to search for datasets using the map. After marking an area on the map and entering search keywords the user expects to receive a list of matching datasets and a representation of these datasets on the map.

It is the task of the NGDS software to execute the geographic and the keyword search and render the information.

Figure 16 shows the interaction between the components.

**sd Map Search**

End User/ Data Consumer

NGDS UI NGDS UI Map

Extension

SOLR Postgres

|  |  |
| --- | --- |
| CKAN Server | |
|  |  |



NavigateToMapPage()

ViewMap()

RenderPage(PageName)

MarkRegionOnMap()

EnterKeywords(Keywords)

GetMarkedRegion()

KeywordSearch(Keywords, region)

SearchForKeywords(Keywords)

FindDatasetsInRegion(list\_of\_datasets, region)

RenderPage(PageName)

RenderPage(PageName)

*Figure 16: MSC for Use Case "Map Search"*

The user initiates map search by navigating to the map search. The NGDS UI will trigger that the map is being rendered by the NGDS UI Map Extension. The user then triggers the map search by marking the search area on the map and by entering keywords into the search bar. When the user presses the search button the NGDS UI retrieves the marked area from the NGDS UI Map Extension and sends the information together with the search keywords to CKAN using CKAN’s default search API. CKAN then uses SOLR to retrieve a list of IDs of all datasets matching the search criteria. It then contact Postgres and uses the PostGIS extension for Postgres to execute a geographic search for all datasets that are overlapping with the marked area. The metadata of all datasets



fulfilling this criterion are returned to the UI for rendering buy the NGDS UI Map Extension.

#### Use Case: “Keyword Library Search”

In this use case the user wishes to search for datasets using keywords and facets. After entering the keywords and activating the relevant facets the user expects to see a list of all datasets fitting the search criteria.

It is the task of the NGDS software to execute the keyword search, filter according to the facets and render the information.

Figure 17 shows the interaction between the components.

**sd Keyword Library Search**

Postgres SOLR

|  |  |
| --- | --- |
| NGDS UI | |
|  |  |

|  |  |
| --- | --- |
| CKAN Server | |
|  |  |

End User/ Data Consumer

EnterKeywords(Keywords)

KeywordSearch(Keywords)

SearchForKeywords(facets, Keywords)

FindMetadata(SearchCriteria)

RenderPage()

*Figure 17: MSC for Use Case "Keyword Search”*

The Faceted Search dynamics simply require that the CKAN server is using SOLR to find the IDs of all datasets that fit the search criteria. It then retrieves the metadata of the fitting datasets from Postgres and returns the information to the NGDS UI for rendering.

#### Use Case: “Upload Shapefile”

In this use case the user creates a new dataset and adds a Shapefile resource to this dataset. The task of the NGDS is to store the resource and associate it with the metadata. It also needs to update the search index so that the dataset can be found using the search mechanisms (Map Search, Keyword Search, Faceted Search). Before it can verify the

Shapefile against a content model and add it to the CKAN data-store it first needs to unpack it so that the individual files within the Shapefile can be processed.

Figure 18 shows the interaction between the components.

**sd Upload Shape File**



Data Submitter

NGDS UI

Postgres SOLR

|  |  |
| --- | --- |
| CKAN Server | |
|  |  |

CKAN Data Store Extension

CKAN Data Store

Shape File Parser

Content Model Verification

CreateDataset(metadata\_name, metadata\_value)

CreateDataset(metadata)

CreateUpdateMetaData(metadata)

CreateUpdateIndexingData(dataset\_id, metadata)

RenderPage(PageName)

CreateResource(metadata\_name, metadata\_value)

CreateResource(datset\_id, metadata)

RenderPage(PageName)

UploadOnlineResourceFile(file\_name)

UploadDataFile(DataFileType, DataFile)

StoreFile()

ParseShapefile(shapfile\_name)

UnzipShapfileAndConsistencyCheck(shapefile\_name)

VerifyFile(shapefile\_name)

UpdateDatstore(shapefile\_name)

ParseFile(shapefile\_name)

StoreData()

StoreDatainNewTable()

RenderPage(PageName)

*Figure 18: MSC for Use Case "Shapefile Upload"*

In many aspects the use case causes the same interactions as uploading structured resources. The difference here is that Shapefile needs to be unpacked before it can be verified and added to the data-store.

#### Use Case: “Harvest Metadata”

In this use case no human user is directly involved. Rather the harvesting process is triggered as a Linux CRON job. At harvesting time the CRON job triggers a process that causes the CKAN harvesting plug-in to start harvesting from a repository. Figure 19 assumes that the repository is a NGDS Node-in-a-Box. It is expected that at the end of the harvesting process all changes within the repository that happened since the last

harvesting time are applied to the aggregated catalog that is maintained by the central harvesting node. In other words, a user must be able to find the datasets in that repository by using the search interface of the central harvesting node.

The task of the NGDS software is to contact the repository using the CSW protocol and to request for a list of all changes since the last harvesting. The Node-in-a-Box must then compile the change list and send it via CSW to the central harvesting node. The list can be very long. The central harvesting node must store the changes in its local catalog.

Figure 19 shows the interaction between the components.

**sd Harvest Metadata**

Central Repository

:Havester

Central Repository

:Postgres

Central Repository

:SOLR

Node-in-the-box

:CKAN Spatial

Node-in-the-box

:Postgres



HarvestTimeExpired()

FindNode(SearchCriteria)

GetUpdate()

FindDatasets()

CreateCatalog()

CreateUpdateMetaData(metadata)

CreateUpdateIndexingData(dataset\_id, metadata)

*Figure 19: MSC for Use Case "Harvest Metadata"*

As can be seen the CKAN Plug-in Central Repository Harvester and its counterpart CKAN Plug-in Spatial Extension do the heavy lifting in this use case. Essentially the harvester contacts the spatial extension over the network. The Spatial Extension compiles the list and returns it to the harvester which applies the retrieved information to both the Postgres database and the SOLR indexer.

#### Use Case “Preview WFS WCS Resources”

In this use case the user wants to preview a resource of type WFS or WCS. The task for the NGDS System is to visually present the WFS/ WCS resource. CKAN come with recline.js support which is able to present structured resources as a table, graph, or on a map (if the structured resource provides geographic information). However, recline does not directly WFS or WCS. However, it does support JSON representations of tables.

Therefore NGDS UI does not access the WFS/ WCS resource directly. It rather accesses it indirectly via translation service provided as a CKAN extension. Figure 20 shows the interaction between the components is presented as a message sequence chart.

**sd Preview OGC Services**



|  |  |
| --- | --- |
| NGDS UI | |
|  |  |

End User/ Data Consumer

Preview WFS(resource, render\_as)

OGC to Json Translator Extension

Geoserver

GetJson(resource)

GetOGCStream()

OGCStream()

JsonObject()

TranslateToJson()

ShowInRecline(JsonObject, render\_as)

*Figure 20: Use Case “Preview WFS WCS Resources”*

Essentially the user initiates the interaction by clicking on the “preview resource” button of a WFS or WCS resource. The UI then calls the RESTful API of the “OGC to JSON Translator Extension”. This extension contacts the OGC server to receive the WFS/ WCS stream. In Figure 20, the OGC server is represented as a Geoserver. However, any server that provides WFS or WCS can be contacted. The data stream from the server is translated into a JSON object which in turn is returned to the UI. The UI then uses its recline.js library to present the WFS/ WCS resource.

### Process Views

The following table provides an overview of all processes triggered by NGDS and how these processes are started.

|  |  |  |
| --- | --- | --- |
| *Process* | *Description* | *Service Type* |
| NGDS-CKAN  core process  CKAN celery damon  NGDS  harvester  NGDS full- text indexer  Postgres Geoserver  SOLR | Runs the NGDS Web Application. Started with:  pas t er ser v er  c on f i gur at i on . i ni  Runs the celery damon which acts as a batch processor for CKAN activities. It runs them in an asynchronous process. When the queue of batch tasks is empty the damon idles.  pas t er cel er y d  Runs the NGDS harvesting process. Started with:  pas t er ha r v es t …  Runs the NGDS full-text indexing service.  Started with:  pas t er f ull t ex t i ndex  Postgres Database  OGC Server. Started with:  s er v i ce j et t y s t ar t  Indexing service. Started with:  s er v i ce j et t y s t ar t | Started as a Linux service. During development strated from the command  Started as a Linux service. During development started from the command line.  Started as a Linux service only on the Central Node.  During development line.  Started with a script. The process dies when there is nothing to index. It is  minutes after it dies last time.  During development started manually from the command line.  Started as a Linux service. Started as a Linux service.  Started as a Linux service |

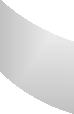
*Table 5-4: NGDS Processes*

### Physical Views

NGDS is a Web Application and physically runs on PC server hardware. The system can be deployed completely on a single machine running all required applications. During development virtual machines (virtual box) with 1 Intel I-7 core, ~1GB RAM and ~20GB disk space are used. For production purposes larger settings are highly recommended. For installation on a single machine the system should provide at least 4 Intel Xeon cores, 8GB of RAM, and sever al hundred MB of disk space.

For performance reasons the system may also be scaled to run on multiple machines. Scaling both horizontally and vertically is possible as is described in the following two Figures.

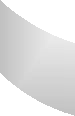
Note that the NGDS development team does not test any of the scaling options.



*Figure 21: NGDS scaled horizontally*

When horizontal scaling is used, the complete system is duplicated on multiple server machines. Each server provides all required instances. The database servers underneath are synchronized using built-in synchronization mechanisms of the database management system. A Load Balancer is required to distribute requests across the machines. NGDS does not have a single-sign-on mechanism so the Load Balancer has to be configured to use sticky sessions.

*Figure 22: NGDS scaled vertically*



Vertical scaling requires running each main component of NGDS on a separate server. This option is likely to be implemented easiest. Some of those components could be duplicated as necessary. Note that vertical scaling does not increase the system reliability since each node in the system is a single point of failure.

## 6 Referenced Materials

|  |  |
| --- | --- |
| KRUCHTEN | Kruchten, Philippe (1995, November). Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50. |
| Barbacci 2003 | Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.  [<http://www.sei.c](http://www.sei.cmu.edu/publications/documents/03.reports/03tr016)m[u.edu/publications/documents/03.reports/03tr016.](http://www.sei.cmu.edu/publications/documents/03.reports/03tr016) html>. |
| Bass 2003 | Bass, Clements, Kazman, *Software Architecture in Practice,* second edition, Addison Wesley Longman, 2003. |
| Clements 2001 | Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001. |
| Clements 2002 | Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002. |
| IEEE 1471 | ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000. |
| REQ-2.7 | Kuhmuench et. al.: NGDS Software Requirements Specification. V2.7.  <http://gdsdpwg.net/NGDSDesign/Shared%20Documents/Requiremen> ts/SCR\_NGDS\_Software\_Requirements\_Specification\_v2.7.docx |
| FEATURES | Bruschi et.al.: NGDS Feature List. <http://gdsdpwg.net/NGDSDesign/Shared%20Documents/Architecture>  /Features.xlsx |
| EPICS | Bruschi, Tanikella et.al.: Prioritization of Epics. <http://gdsdpwg.net/NGDSDesign/Shared%20Documents/Sprints/Prior> itization%20of%20Epics.xlsx |
| PYLONS | James Gardner: The Definitive Guide to Pylons. 2009. ISBN 1- 59059-934-9 |

## Di rector y

### Glossary

|  |  |
| --- | --- |
| *Term* | *Definition* |
| software architecture  view  view packet  viewpoint | The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible” properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.  A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.  The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.  A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. |

* 1. **Acronym List**

|  |  |
| --- | --- |
| *Acronym* | *Definition* |
| API  ATAM CMM CMMI CORBA COTS EPIC IEEE KPA OO ORB OS QAW RUP SAD SDE  SEE | Application Programming Interface; Application Program Interface; Application Programmer Interface  Architecture Tradeoff Analysis Method Capability Maturity Model.  Capability Maturity Model Integration Common object request broker architecture Commercial-Off-The-Shelf  Evolutionary Process for Integrating COTS-Based Systems Institute of Electrical and Electronics Engineers  Key Process Area Object Oriented Object Request Broker Operating System  Quality Attribute Workshop Rational Unified Process Software Architecture Document  Software Development Environment  Software Engineering Environment |

SEI Software Engineering Institute, Systems Engineering & Integration, Software End Item

SEPG Software Engineering Process Group

SLOC Source Lines of Code

SW-CMM Capability Maturity Model for Software

CMMI-SW Capability Maturity Model Integrated - includes Software Engineering

UML Unified Modeling Language

NGDS National Geothermal Data System

CKAN Comprehensive Knowledge Archive Network DOE Department of Energy

GDR Geothermal Data Repository

AZGS Arizona Geological Survey

WFS Web Feature Service

WMS Web Map Service

WCS Web Coverage Service

CSW Catalog Service For the Web

OGC Open Geographical Consortium

OKFN Open Knowledge Foundation

SAD Software Architecture Document