OpenSpending

Functional Specification

Document Changes:

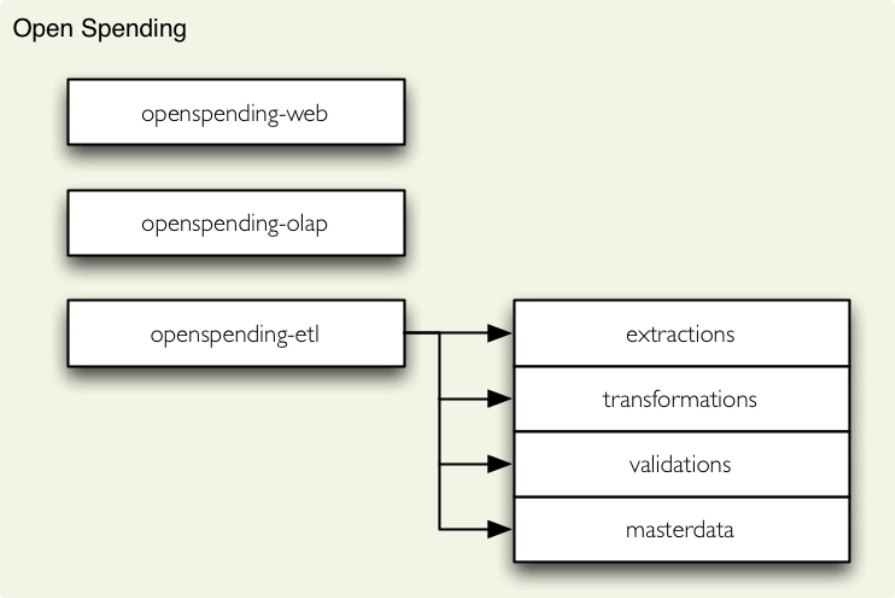
Date	Author	Changes
2011-07-15	stiivi	Added information about ETL

Contents

Proposals

Architectural changes

- legacy application package wdmmg and wdmmg-ext will be split into more logical and functionally coupled packges:
    - OpenSpending Web - web front-end application, with interface for pluggable visualisations
    - OpenSpending ETL - extraction, transformation and loading package
    - OpenSpending OLAP - online analytical processing package



- main OpenSpending packages should focus on "core business" - that is providing analytical insight into spending data, either through web based interface, search engine or API.
- analytical dataset is read-only for web application

Changes required:

- remove non-web code from the wdmmg package. ***Reasons***: cleaner packages, easier to maintain
- separate raw/analytical data layer: well defined API to get analytical and raw data from data backend – ***Reasons***: more transparent architecture, easier to maintain and track-back bugs (software or data); more architectural flexibility – easier to make changes; easier to introduce new analytical functionality

- web application should not depend on data structures and physical data storage, all queries should go through raw/analytical data API – ***Reasons***: future changes require less work, are less prone to errors; scalability is matter of backend not of application
- remove dataset specific code and all dataset specific extraction, transformation and loading code should be delegated to third-parties. ***Reasons***: keeps code clear; separation forces to use encapsulated API oriented architecture, which is more transparent and easier to maintain; encourages reusability of ETL code

Data sources and storage changes

New functionalities:

- master management will be introduced - process for creation, maintenance, update of lists, enumerations, dimensions, classifications, mappings, etc.
- ETL process monitoring will be introduced – all ETL jobs should be auditable through single place

Changes:

- split current application data store into multiple stores depending on data stage: source mirror, staging area/operational data store, datamart area, application specific data (described later). ***Reasons***: transparent data processing process; modular architecture wich is easier to maintain; provides basis for better data quality – data quality issues are easier to track; open for scalability at any given level without need to rewrite the rest of the OpenSpending system(s)
- each dataset in OpenSpending should have a corresponding CKAN package – ***Reasons***: transparency of data sources; availability to third parties; crowd-sourced source data auditability; last but not least (non-openspending related) - encourages use of CKAN and serves as use-case example of CKAN usage
- most of master data sources (classifications, lists, enumeration) should be available as CKAN packages as well, mainly list of entities, classfications – ***Reasons***: same as reasons for dataset source being stored in CKAN; introduces better reusability of classifications at source level – potential data providers can be pointed to open and public existing classifications to make their data comply with OpenSpending requirements.

Restrictions:

- no direct data change in database should be done manually, every change should be introduced in ETL process, even small bug fixes – ***Reasons***: transparency; better data quality; data reconstructability
- whole data mart should be regenerated by running required ETL scripts – ***Reasons***: in case of data loss (database attack, hacking, manual changes), whole datamart can be regenerated to the same state as it was before just using scripts

Architecture Proposal 1

Top-level modules from user's level down to data level:

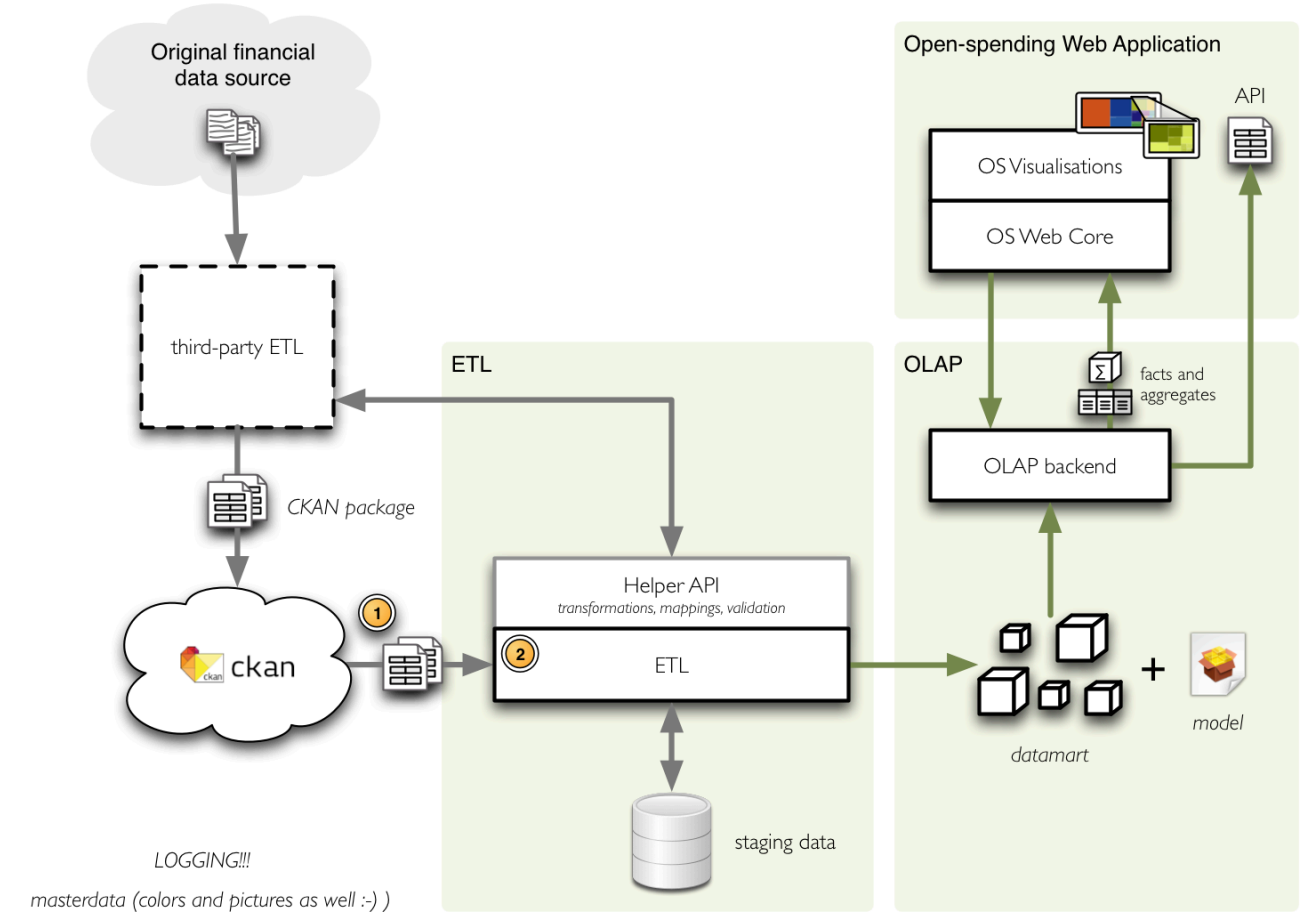
- OpenSpending Web Application
- Online Analytical Processing
- ETL

External modules and dependencies:

- CKAN – not part of OpenSpending, but required as data source
- third-party ETL packages – created by *"data wranglers"*

The architecture is depicten in the following diagram:

OpenSpending – Modules (high level)



OpenSpending Web Application

- provides web interface for visualising, browsing and searching OpenSpending datasets
- has notion of "reports" – description of how datasets are being visualised and analysed,

which dimensions, measures and post-computations are used
- might contain application data store for storing "transactional application data", such as comments, taggings, customized reports
- web application accesses analytical data in read-only mode

Online Analytical Processing (OLAP) Module

- provides interface for querying analytical data: from facts to aggregates
- has well-known read-only interface which serves as the only way for accessing open-spending datasets from the end-user web aplication

Interface should provide functionalites:

- retrieve detailed information about spending (fact/detail)
- retrieve metadata describing a dataset entries
- retrieve list of dimension entries (such as list of entities, list of classifications, ...)
- aggregate measures through dimensions

API responses should be in the most appropriate format:

- Python objects when called from python
- JSON when called using HTTP API
- CSV through HTTP where appropriate

There might be possible future analytical capabilities of the OLAP backend:

- pivot tables – cross-table structures with dimensions in rows and columns, possibly paginated by another dimension; should be provided in the most appropriate form that can be immediately used with result consumer without any major transformation (for example: JSON reply should contain list of row and column titles and then table cells as list of rows where row is list of column cells, already sorted according to pivoted dimensions)
- histograms
- pre-computation of measure-based classification dimensions (for example: low/medium/high expense)
- association mining (apriori/shopping basket) – example use from spending data: "what combination of subjects of contracts is most common?"
- etc.

Note: search engine might or might not be part of OLAP module, recommended is indexing out-of OLAP module with references to OLAP objects (multidimensional aggregates, detailed facts)

Data Storage

Implementation of data storage for OLAP should be the most appropriate with regard to:

available software, knowledge of developers, maintainability, auditability and ease of use for analytical processing.
The OLAP data store should be abstracted and hidden from application – no direct access should be allowed, analytical queries should be done only through analytical API. Abstraction makes architecture cleaner, easier to maintain and more transparent, therefore more auditable. Also is open for future changes, such as decision to change analytical store for scalability or feature richness reasons.
There are two actual possibilities of data store backends:

- relational database (ROLAP)
- mongo-db

Preference is for ROLAP for following reasons:

- high availability of knowledge and best practices for ROLAP, well tuned schemas and processes
- very easy to implement
- SQL is very good for ad-hoc queries – allows wide variety of fast ad-hoc reports on top of structures created by ROLAP backend

References:

- ROLAP: http://en.wikipedia.org/wiki/ROLAP

Extraction, Transformation and Loading

Purpose of the ETL package is to provide functionality to perform source data extraction, validation, transformations and final loading into the analytical store.

Changes

- ETL process is to be split into multiple steps, depending on data stage. *Reason*: Auditable process
- ETL is encapsulated module with API, application will have no access to the internal structures of ETL data, except operational data store

New features

- master data management process
- unified ETL logging

Terminology

- fact dataset – collection of records with financial transactions
- dimension dataset – collection of records with dimension information, such as classification
- source file – partial

Reserved words (to avoid confusion):

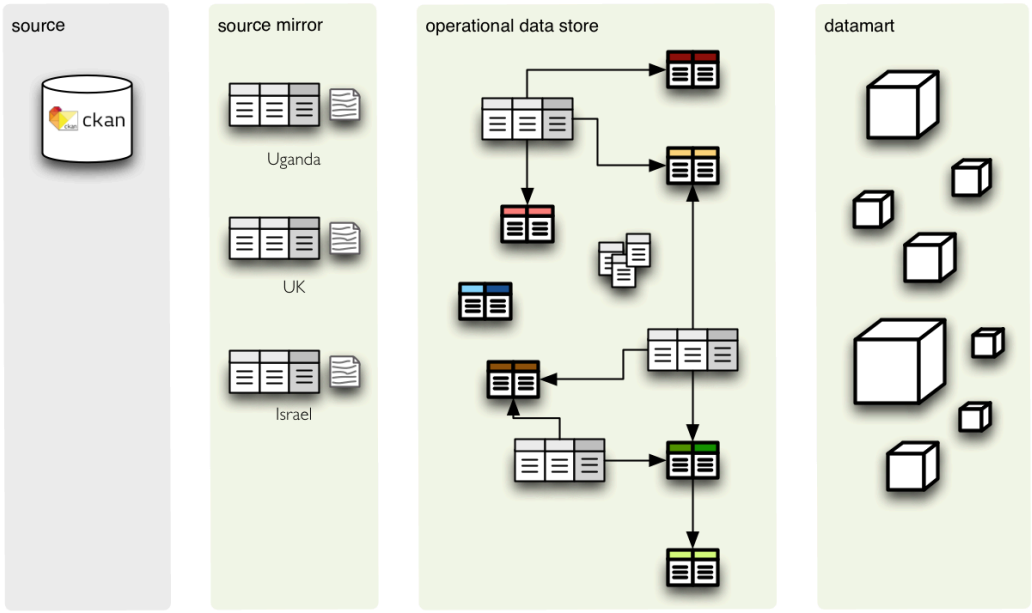
- partition – PostgreSQL table partition
- slice (OLAP) – cell from multidimensional cube

Data Storage Areas

Data are being store in multiple forms in different areas, depending on processing stage. The areas are:

- source mirror – exact copy of data resource with metadata
- operational data store – contains transformed, pre-processed data, can be used for ad-hoc reporting
- analytical data store (OLAP) – data in analytical form for multidimensional aggregate browsing

OpenSpending – Data Areas



In addition to data storage areas, ETL process will require separate storage space (SQL schema or mongo DB collection) to store control information.

Source Mirror

The source mirror contains exact copy of data resources accompanied by resource metadata and data. Preferred way of storage is file system - most portable, immediately accessible, auditable without special tools.
Granularity of the source mirror is dataset

Operational data store

Contents:

- assembled fact datasets from source files – each dataset collection contains all transactions from a dataset
- lists – base for dimensions
- mappings – tables for key-based mapping to derive new fields

Preference for ODS backend is SQL, for the following reasons:

- more readable and maintainable transformations code
- faster mappings
- easier ad-hoc reporting

Disadvantages of SQL compared to noSQL:

- requires good and reliable metadata management
- difficult to handle frequently changing structures

Analytical Data Store

The data in store for OLAP will be stored in the form most appropriate for the aggregated browsing. Data structure should be encapsulated and hidden from "outside world". Application should access the analytical data only throught well defined API.

ETL Processes

Process	Description	Input	Output
Update source metadata	Load metadata about source packages from CKAN and store them locally.	List of CKAN packages: dataset name, package name (if not set, then equal to dataset name), dataset type (facts or dimension)	JSON record per package
Download Source	Create local mirror of source files	dataset source package information	Downloaded source file (in the Source Area)
Validate Source File	Validate source file whether it matches metadata specification and basic quality checks	dataset source package information, source file	validation report
Load dataset file	Loads a dataset file into ODS. If the file was already loaded, it will be replaced.	dataset source file	records in dataset ODS table
Transform dataset	Performs field based transformations on datsaets, for example: derive measures, consolidate currency, derive fields	ODS dataset	ODS dataset

	based on mappings, ... This stem might contain dataset-specific modules.		
Update dataset snowflake	Updates dataset snowflake by mapping dimension datasets, such as classifications or entities. This step should be generic for every dataset.	ODS dataset source	ODS dataset facts
Create/update cube	Transforms dataset into multidimensional analytical format.	ODS snowflake	OLAP cube

Dataset lifecycle

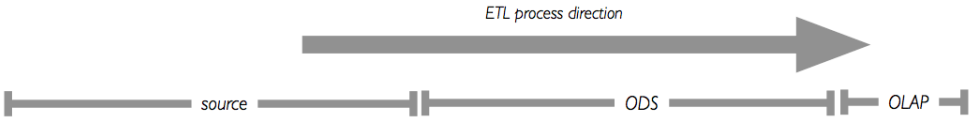
While dataset processing, either fact or dimension, status information should be stored in ETL control structures.
The dataset has multiple life-cycle stages, with one special "pre-natal" stage for source files, which is handled separately.
Dataset stages:

- metadata – only dataset information is known, no structures are prepared
- structures – structures are prepared, no data is loaded
- source – source files are loaded and assembled into a dataset
- transformed – transformations passed
- mapped
- snowflake
- cube

Each dataset stage should have information:

- date created
- stage status – empty, ok, failed
- failure reason

Following image shows exaple of different dataset statuses:



Dataset	metadata	structures	source	validation	transformation	snowflake	cube
uk	✓	✓	✓	✓	✓	✓	✓
israel	✓	✓	✓	?	?	?	?
italy	✓	✓	✓	✗	?	?	?
germany	✓	✓	✗	✗	✓	✓	✓
hungary	?	?	?	?	?	?	?

UK – dataset loaded, working correctly, available in application
israel – new dataset, source loaded, not yet validated
italy – new dataset, source loaded, validation failed
germany – dataset available in application, working correctly; loading of new data failed – some sources were not downloaded, validation failed
hungary – new dataset, just entered to the system, metadata from CKAN yet to be loaded