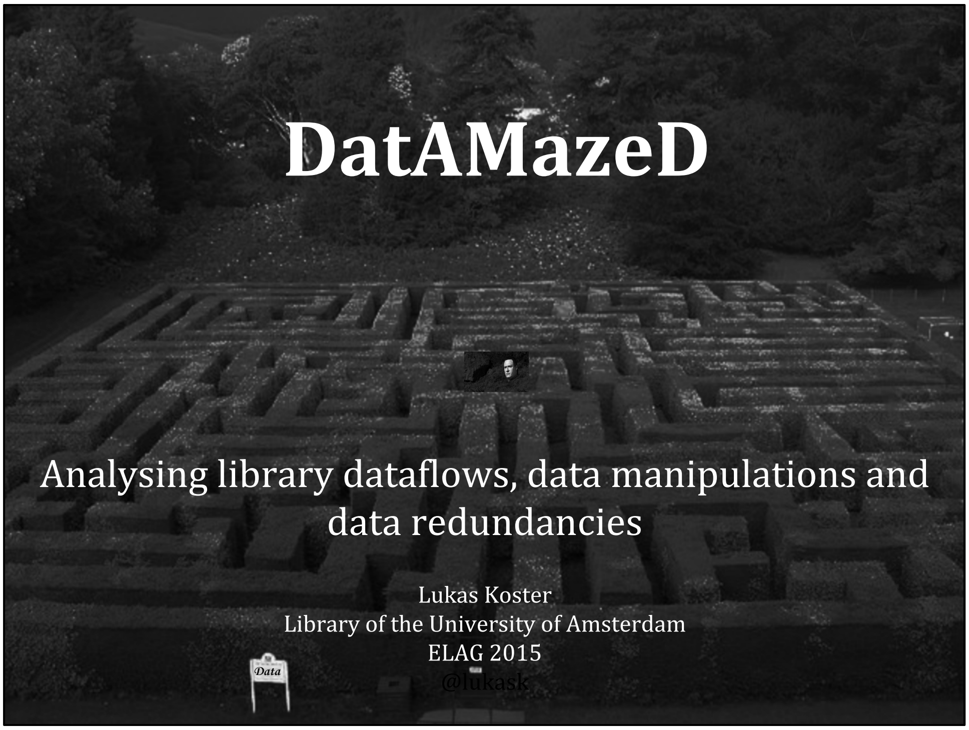
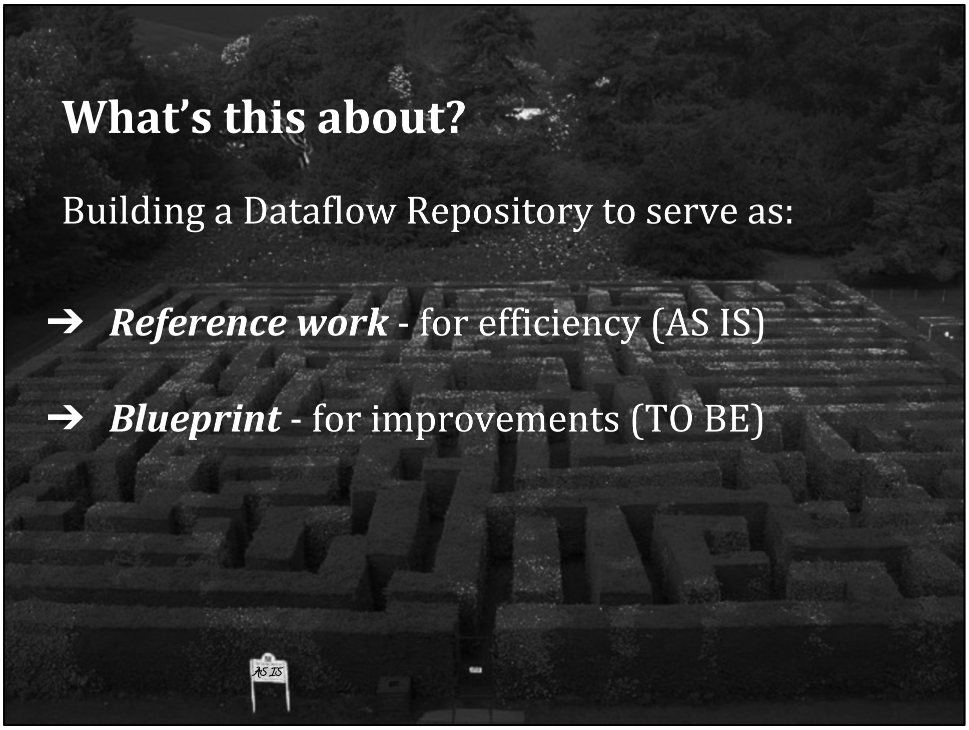


DatAMazeD

Analysing library dataflows, data manipulations and data redundancies

Lukas Koster
Library of the University of Amsterdam
ELAG 2015
@lukask

Original title contains the term “data manipulations”. I actually came to prefer the more neutral term “data transformations”, but I left it here on the title slide in order not to confuse the organizing committee :-)



What's this about?

Building a Dataflow Repository to serve as:

➔ ***Reference work*** - for efficiency (AS IS)

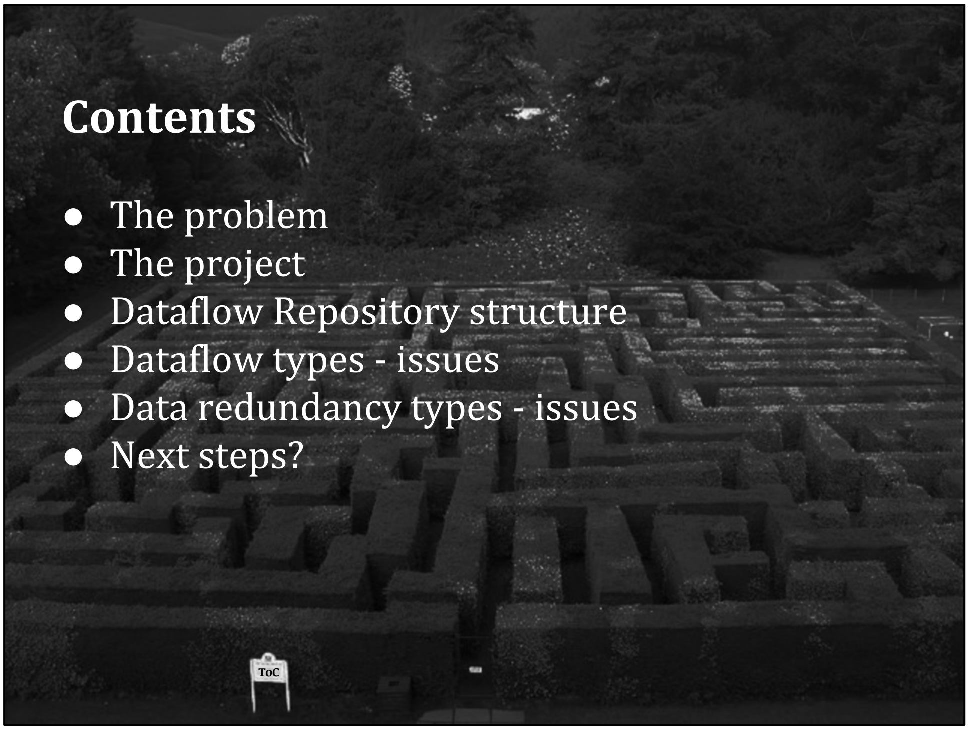
➔ ***Blueprint*** - for improvements (TO BE)

This presentation is about the Library of the University of Amsterdam project "Dataflow Inventory".

The project's objective is to develop a Dataflow Repository serving as:

○ Reference work for (meta)data types/formats, object/content types, flows, system dependencies (Current infrastructure: 'AS IS')

○ Blueprint - Starting point for efficiency improvements, innovative services, system independent data exchange (Future infrastructure: 'TO BE')

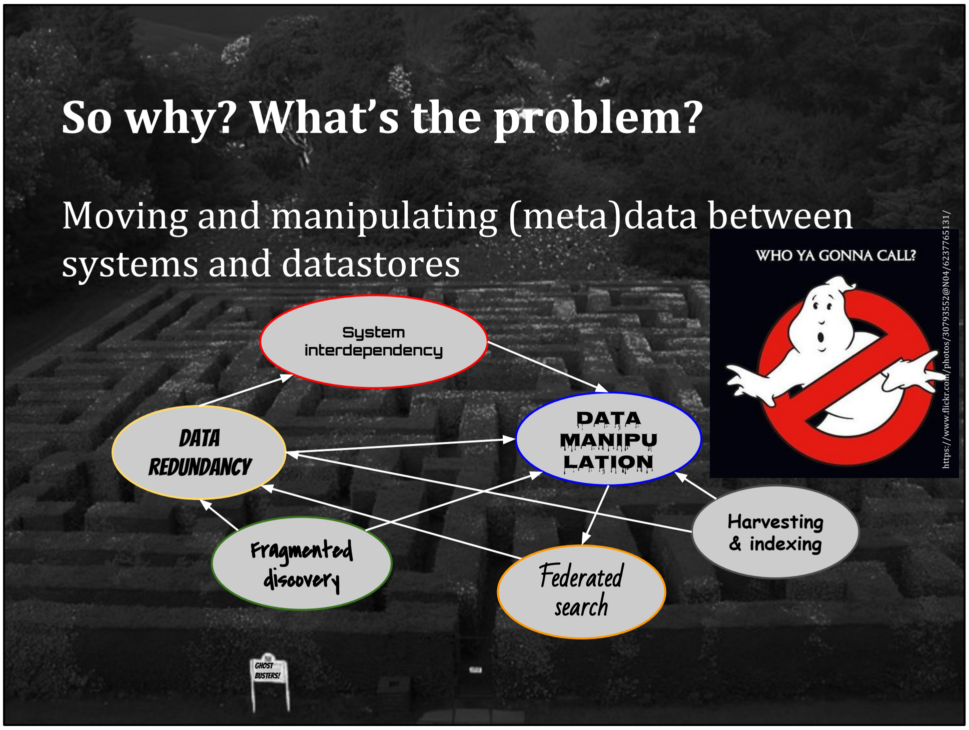


Contents

- The problem
- The project
- Dataflow Repository structure
- Dataflow types - issues
- Data redundancy types - issues
- Next steps?

Contents of this presentation:

- Description of the problem and the background
- Description of the project, methodology, tool
- Description of the structure of the repository using the methodology and the tool
- Types of dataflows and their issues
- Types of data redundancy and their issues
- Benefits and possible next steps

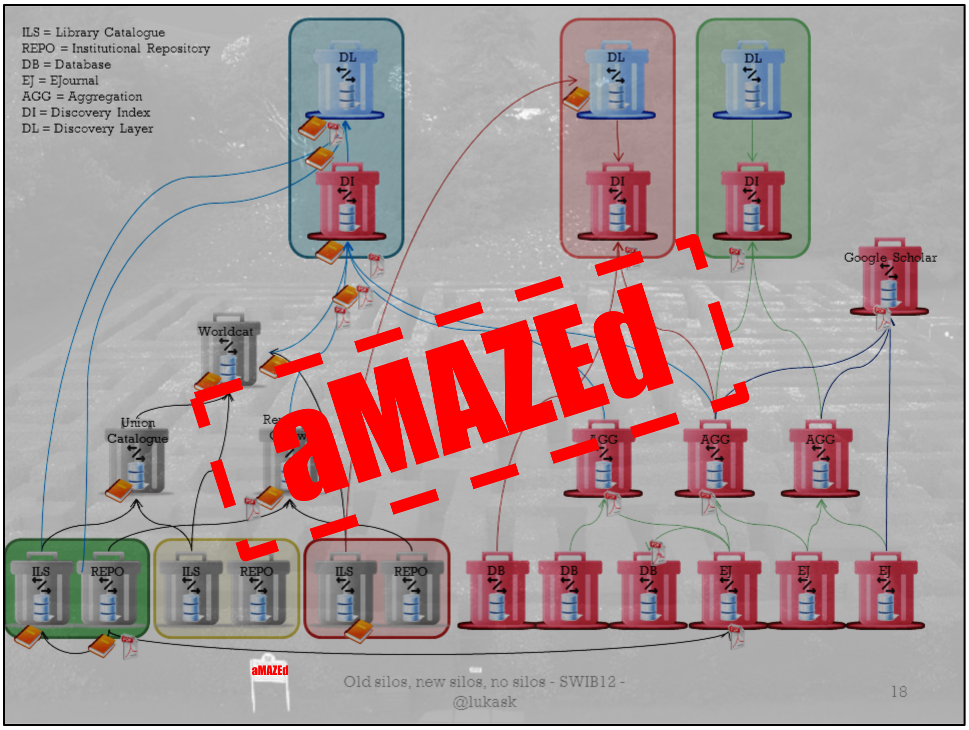


So why? What's the problem?

Moving and manipulating (meta)data between systems and datastores

https://www.flickr.com/photos/30793552@N04/6237765131/

The ultimate goal of the project is to improve the information infrastructure of the library.
Huge amounts of time and effort are spent on moving data from one system to another and shoehorning one record format into the next, only to fulfill the necessary everyday services of the university library.
This creates critical system interdependencies and proprietary data transformation configurations.

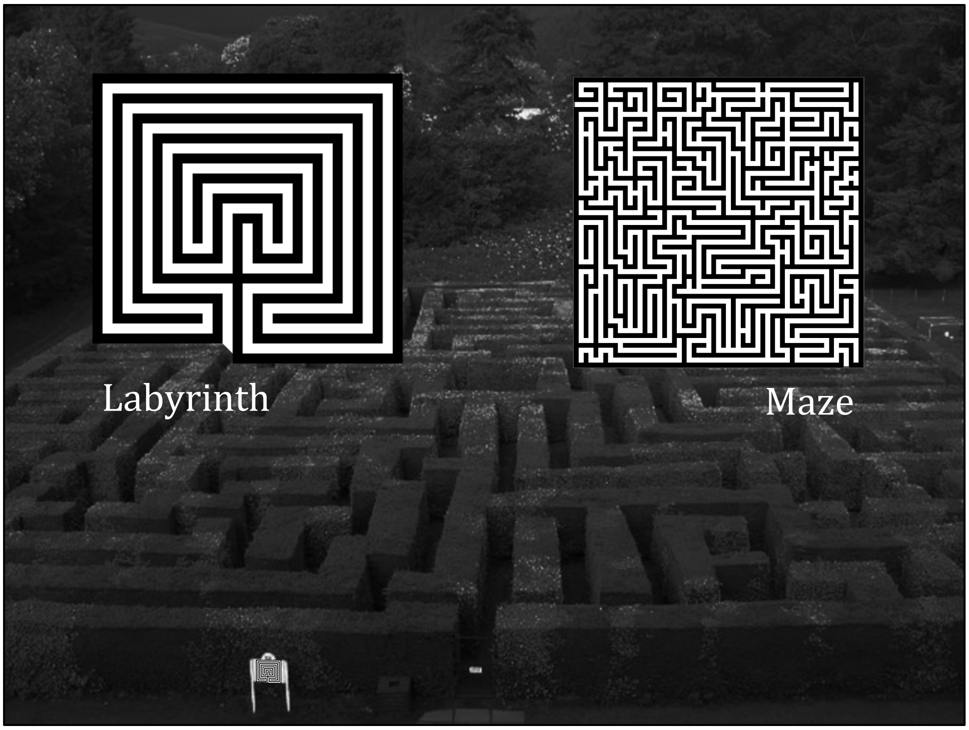


ILS = Library Catalogue
REPO = Institutional Repository
DB = Database
EJ = EJournal
AGG = Aggregation
DI = Discovery Index
DL = Discovery Layer

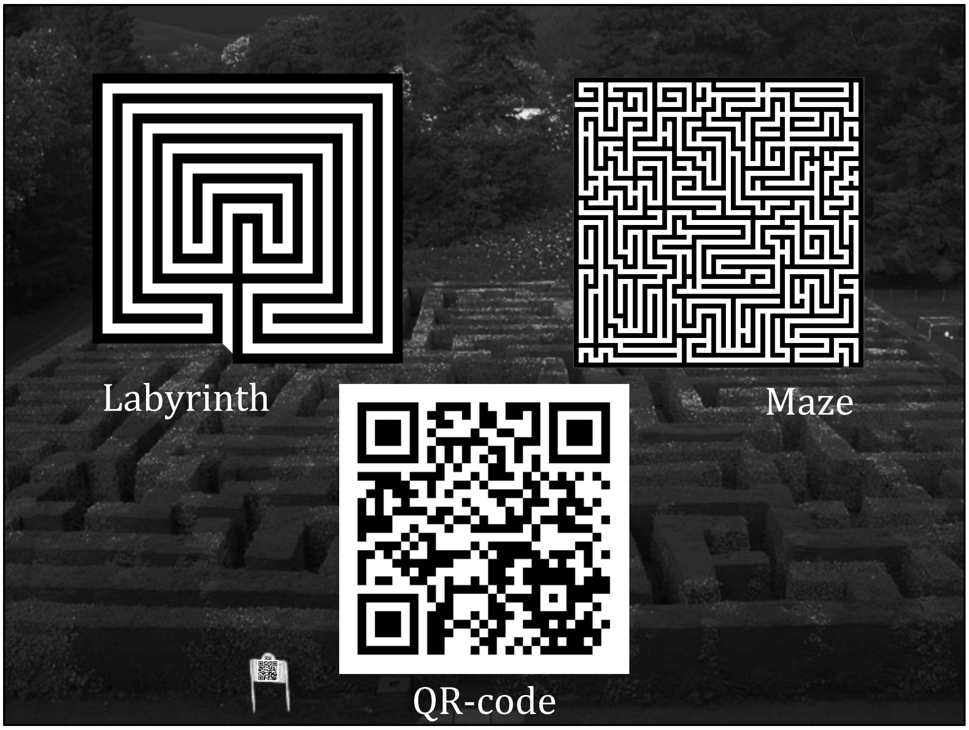
Old silos, new silos, no silos - SWIB12 - @lukask

18

It’s also the cause of a lot of data redundancy.
When drawing this overview of metadata distribution for a SWIB12 presentation, I was really aMAZEd how far and wide one item’s data is dispersed.
The word “amazed” is etymologically related to “maze”, which this diagram looks like.



There is a difference between a maze and a labyrinth. At least according to some experts.
A labyrinth has a single through-route with twists and turns but without branches.
A maze is a confusing pathway that has many branches, choices of path and dead-ends.
Library data infrastructure resembles a maze, not a labyrinth.



And then there's the QR code!

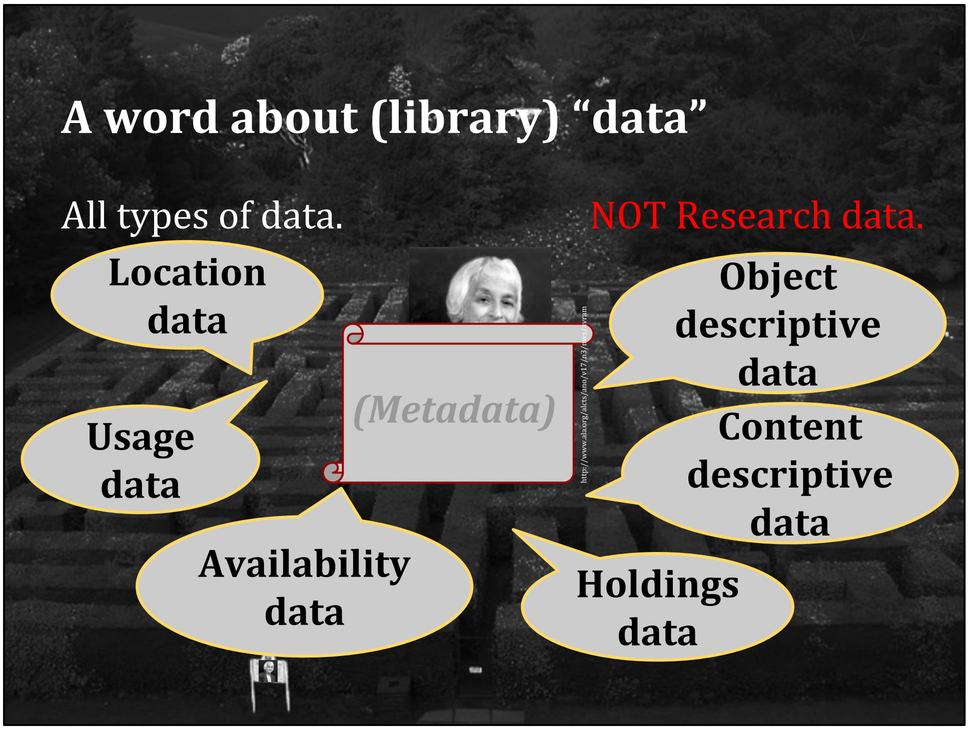
https://www.flickr.com/photos/thomas-merton/2989653563

So we’re facing a Data Maze, where you can easily get lost.
Not only end users of the library, but also library staff: cataloguers, data experts, systems people, domain experts

“Not only is it not possible to invest this time and effort productively in innovative developments, but this fragmented system and data infrastructure is also completely unsuitable for fundamental innovation. Moreover, information provided by current end user services is fragmented as well. Systems are holding data hostage.”

Dependency on systems provided by one or a few companies is sometimes referred to as “vendor lock-in”, something that we need to avoid, we are told. In reality however, there is not much else. We are always dependent on one or more system and database providers (not only commercial ‘vendors’, but also local developers and free and open source software communities). Better to speak of “systems lock-in” (also referred to as “silos”).
Anyway, from a system management and efficiency perspective ‘vendor lock-in’ appears to be better than the chaotic multi system and silo environment. This does not mean that you won’t have any problems, but you don’t have to solve them yourself. From the innovation perspective however, this is not the case. But in my view there is not much difference here with a fragmented infrastructure.

It would however be great if we can free the data, in such a way as to minimalize (not eliminate) these dependencies, which would also lead to more efficient and innovative investment of people, expertise and funding.



A word about (library) "data"

All types of data.

NOT Research data.

Location data

Usage data

Availability data

(Metadata)

Object descriptive data

Content descriptive data

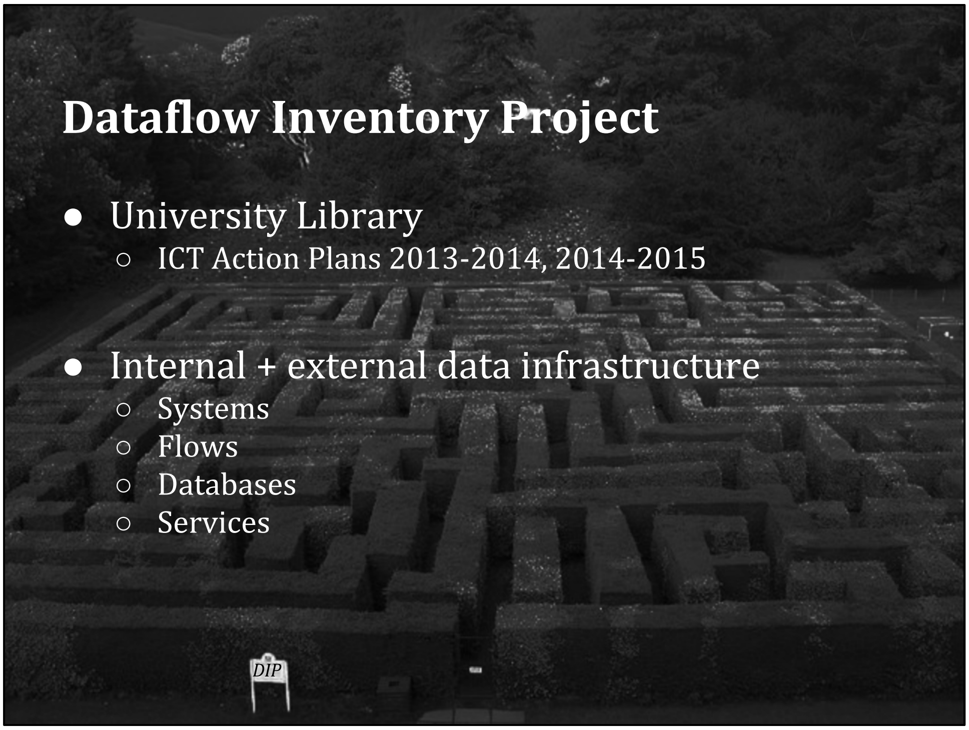
Holdings data

http://www.ala.org/alcts/ano/v17/n3/nws/avram

With "(library) data" I do not mean "research data" (as is used more and more recently in 'data librarian' etc.), nor just what is commonly referred to as "metadata". I don't like to use the term "metadata". It's also not just "bits" (like in telecom provider talk, or elementary information science).

In library environments we have various types of data: object descriptive data (physical format, dimensions, etc.), content descriptive data (subjects, periods, etc.), location data (where is the object located), availability data (how can I get access to it), usage data (how much, by who is it used), etc.

In this specific case "research datasets" are merely one type of content/object.

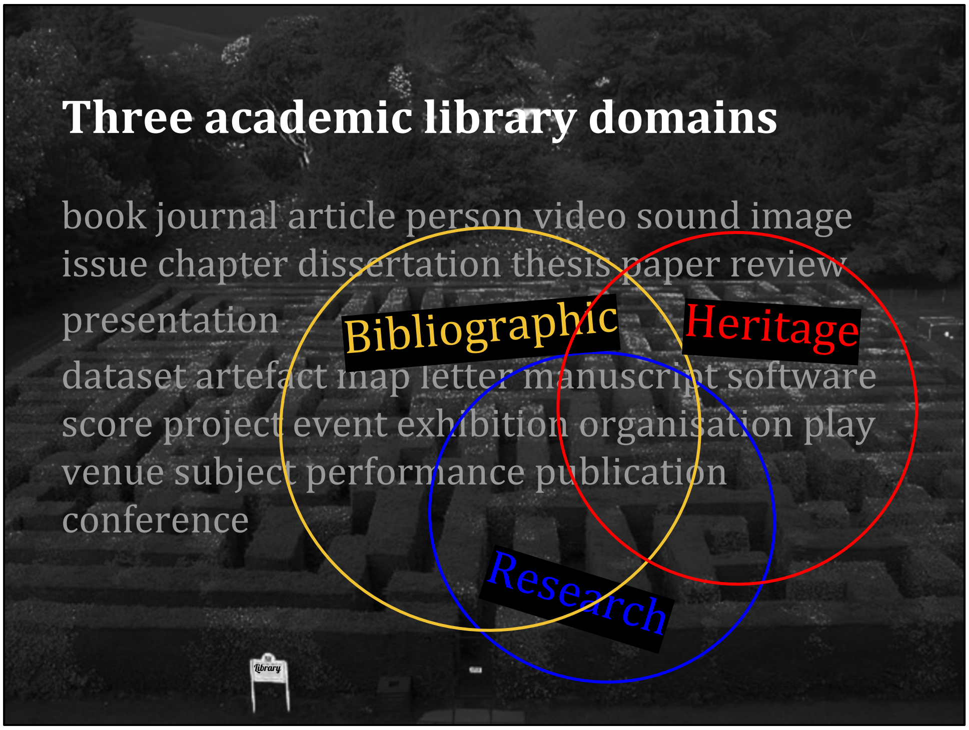


Dataflow Inventory Project

- University Library
  - ICT Action Plans 2013-2014, 2014-2015
- Internal + external data infrastructure
  - Systems
  - Flows
  - Databases
  - Services

One of the umbrella projects/programmes in the University Library short term ICT Action plans: "Development and improvement of the information architecture and data flows"
Within this context the Dataflow Inventory Project was started, because first of all "a thorough analysis of a library's current information infrastructure is required".

"The goal of the project is to describe the nature and content of all internal and external datastores and dataflows between internal and external systems in terms of object types (such as books, articles, datasets, etc.) and data formats, thereby identifying overlap, redundancy and bottlenecks that stand in the way of efficient data and service management."



Three academic library domains

book journal article person video sound image issue chapter dissertation thesis paper review presentation

dataset artefact map letter manuscript software score project event exhibition organisation play venue subject performance publication conference

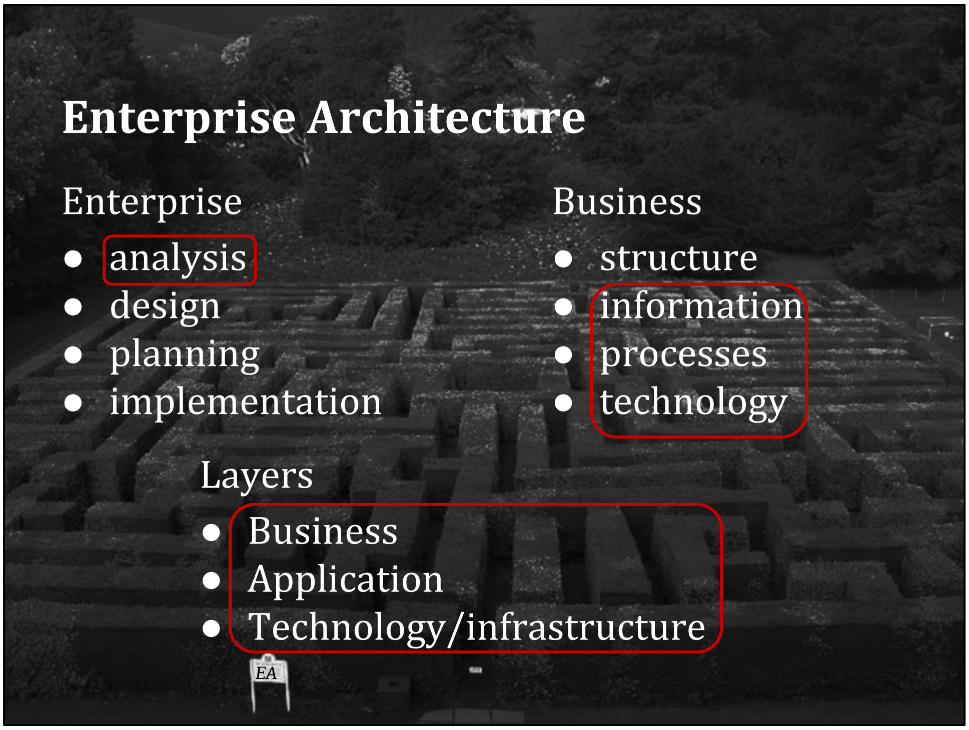
The Library of the University of Amsterdam, as many academic libraries, is managing data for three ‘domains’, Bibliographic (traditional library), Heritage, Research. Content and data in these three domains largely overlap, but there are also differences in types and services/customers.

The domains can be regarded also as three different perspectives on library data.

So, how do we do this?

- Method
- Methodology (yeah yeah)
- Tools
- Other libraries/similar projects

What is needed to make an inventory of dataflows in an academic library?
A method (how do we do it) and a methodology (which conventions do we use). Yeah, there is a difference.
With what can we do it? Which tool suits our needs and the chosen methodology best? And is not too expensive?
We don’t want to invent the wheel: have other libraries done similar projects?



Enterprise Architecture

Enterprise

- analysis
- design
- planning
- implementation

Business

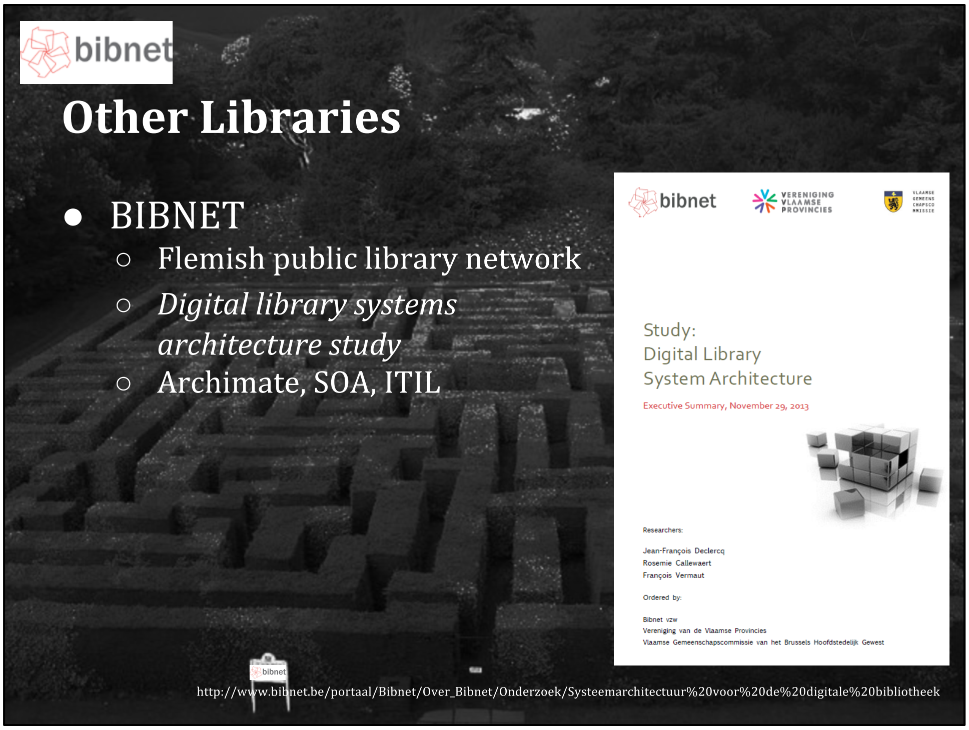
- structure
- information
- processes
- technology

Layers

- Business
- Application
- Technology/infrastructure

Since the initial and primary goal of this project is to describe the existing infrastructure instead of a desired new situation, the first methodological area to investigate appears to be Enterprise Architecture: "...*practice for conducting enterprise analysis, design, planning, and implementation*..." - "...*guide organizations through the business, information, process, and technology changes*..."
(27 november 2014: Wikipedia states "*This article appears to* ***contain a large number of buzzwords***". Not anymore.)

"Enterprise"= "the organization" (more or less).
"Business"= "the services delivered"(more or less).
In our project we need "analysis" (from the Enterprise side) of "information" "processes" and "technology" (from the Business side).
Two main EA frameworks/methodologies used globally. TOGAF, Archimate (which complies to TOGAF).
Three main areas/levels: Business, Application, Infrastructure.
Exactly what we need. Service areas (Business Layer), Systems (Application Layer), Databases/Platforms (Infrastructure). With some overlap of course (databases).



Other Libraries

- BIBNET
  - Flemish public library network
  - *Digital library systems architecture study*
  - Archimate, SOA, ITIL

Study:
Digital Library
System Architecture

Executive Summary, November 29, 2013

Researchers:

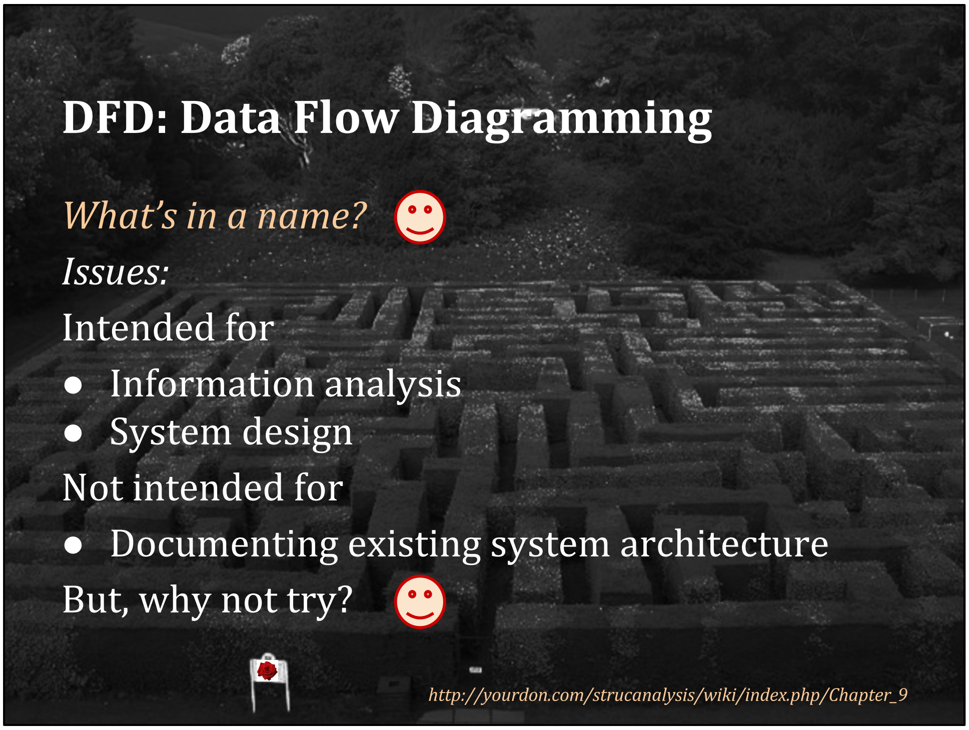
Jean-François Declercq
Rosemie Callewaert
François Vermaut

Ordered by:

Bibnet vzw
Vereniging van de Vlaamse Provincies
Vlaamse Gemeenschapscommissie van het Brussels Hoofdstedelijk Gewest

http://www.bibnet.be/portaal/Bibnet/Over_Bibnet/Onderzoek/Systeemarchitectuur%20voor%20de%20digitale%20bibliotheek

Trying to find similar work, we knew of BIBNET Flemish Public Library Network and their Architecture Study. Contact Rosemie Callewaert.
They focused on the big picture, not the dataflows as such. Using Archimate among others.



DFD: Data Flow Diagramming

What's in a name? ☺

Issues:

Intended for

- Information analysis
- System design

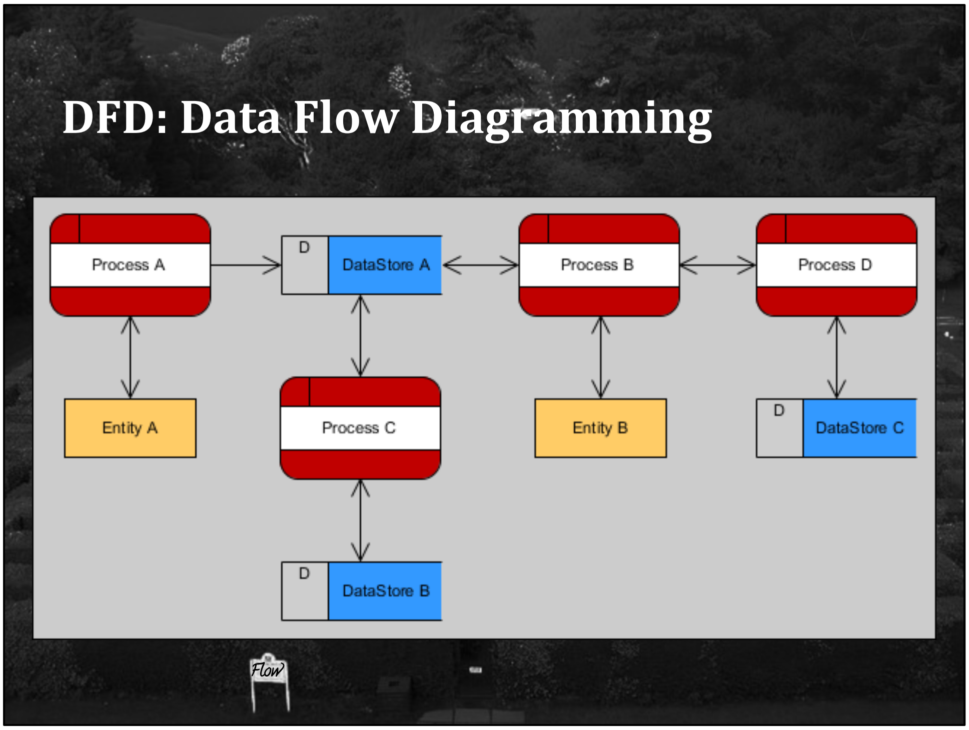
Not intended for

- Documenting existing system architecture

But, why not try? ☺

http://yourdon.com/strucanalysis/wiki/index.php/Chapter_9

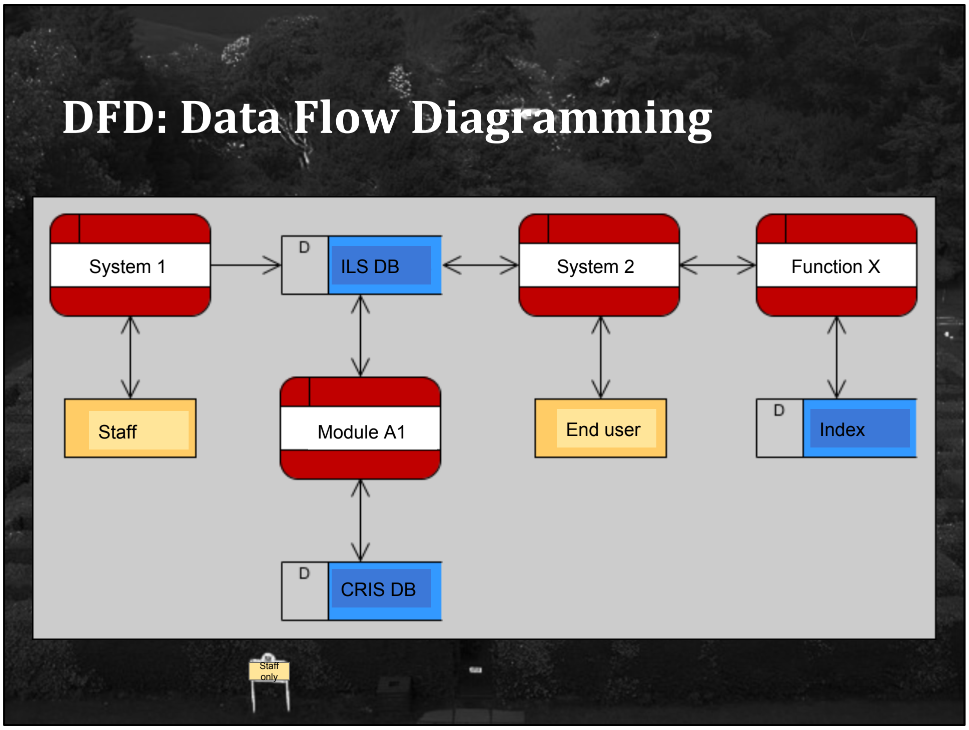
Looking at tools, a number of packages provide a number of methods/methodologies for Business Process Modeling, UML, System Design, etc.
An old method(ology) is Data Flow Diagramming (DFD), which I knew from systems designer days.
Part of the Structured Analysis Methodology (Yourdon). Structured Analysis/DFD is used for analysing business workflows, information flows etc. in order to design systems. We decided to adapt the method for describing already existing systems and information architecture, because it is relatively easy to use (compared to methods like Archimate) and it would fit our initial purpose.



DFD: Data Flow Diagramming

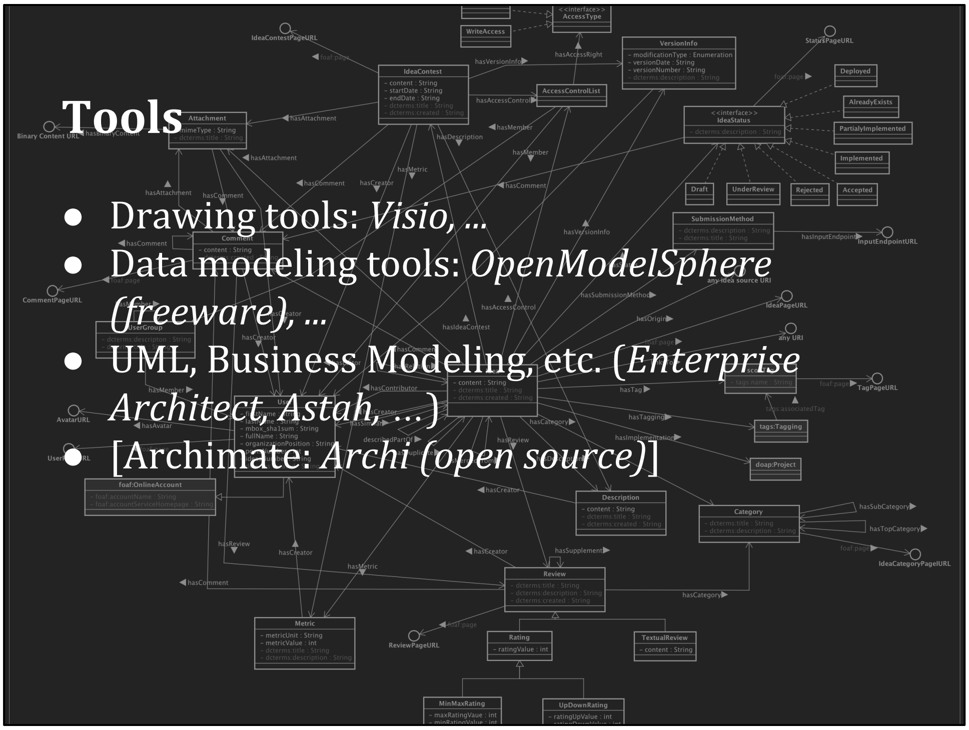
Example of traditional DFD: describing information architecture on a conceptual/real life level

For instance: Entity A=Librarian, Process A=Cataloguing, Datastore A=Publication records



DFD: Data Flow Diagramming

Our adaptation of DFD’s: describing information architecture on an implementation level, where Process are Systems, Modules or Functions.
System 1 could for instance be an ILS, like Aleph.



Tools

- Drawing tools: *Visio, …*
- Data modeling tools: *OpenModelSphere (freeware), …*
- UML, Business Modeling, etc. (*Enterprise Architect, Astah, …*)
- [Archimate: *Archi (open source)*]

We needed a repository based tool which enables reuse of elements in other diagrams, linking, descriptions, reporting.

Basically there are three types of tools that can be used for diagramming.
 Drawing tools, cons: no reuse of elements, reports
 Data modeling tools, cons: mainly focused on data and relationships, not flows and processes
 Business/system modelling, with all kinds of method(ologie)s.

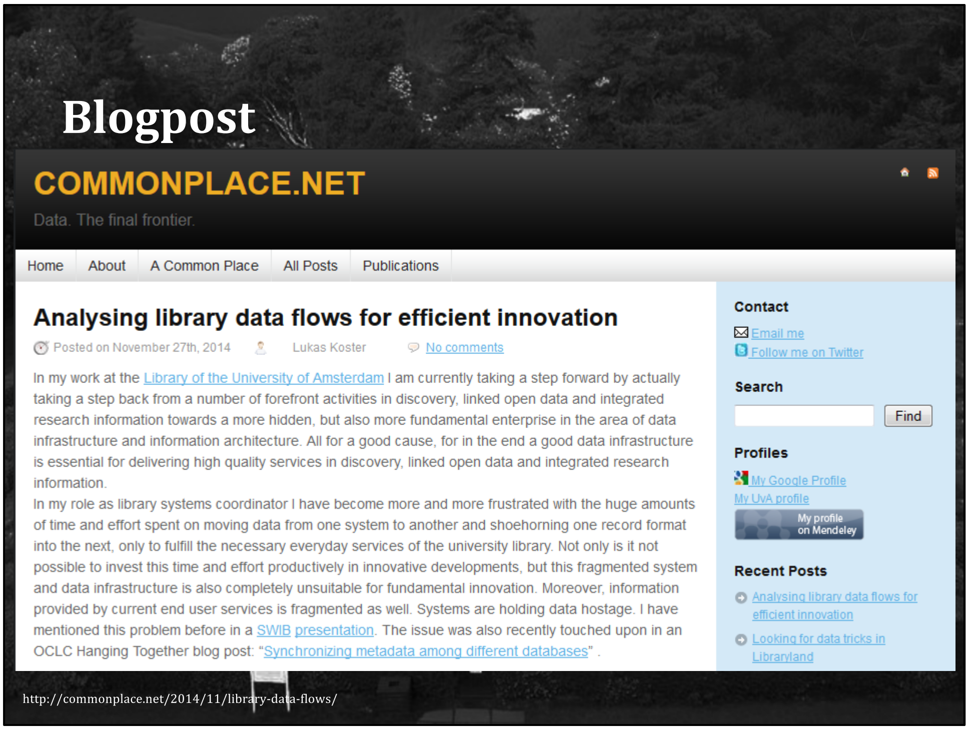
Tool

Visual Paradigm

- DFD (and other methods)
- Glossary/Dictionary
- Export/Import
- Shared repository
- Documentation
- Reports

We chose Visual Paradigm because of these reasons + the costs.
The tool is relatively open, with a number of export/import formats, a lot of reporting and documenting options, a shared online multi user version management model repository, and the option to use it for other purposes with other methods in the future.
Company and support is very responsive.

Blogpost



Analysing library data flows for efficient innovation

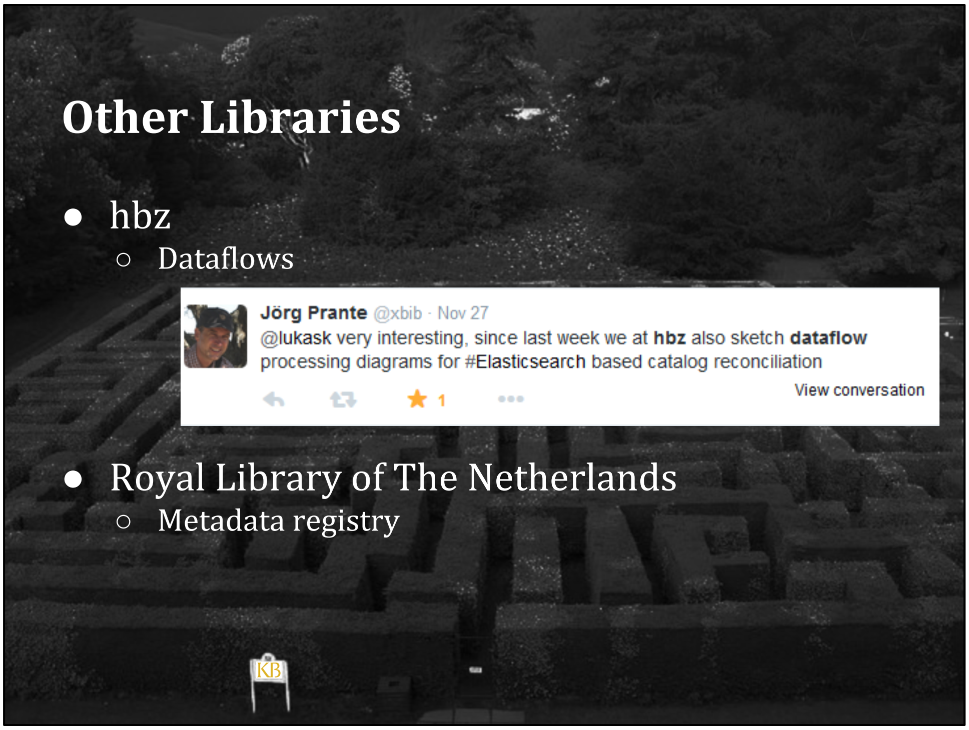
Posted on November 27th, 2014 Lukas Koster No comments

In my work at the Library of the University of Amsterdam I am currently taking a step forward by actually taking a step back from a number of forefront activities in discovery, linked open data and integrated research information towards a more hidden, but also more fundamental enterprise in the area of data infrastructure and information architecture. All for a good cause, for in the end a good data infrastructure is essential for delivering high quality services in discovery, linked open data and integrated research information.

In my role as library systems coordinator I have become more and more frustrated with the huge amounts of time and effort spent on moving data from one system to another and shoehorning one record format into the next, only to fulfill the necessary everyday services of the university library. Not only is it not possible to invest this time and effort productively in innovative developments, but this fragmented system and data infrastructure is also completely unsuitable for fundamental innovation. Moreover, information provided by current end user services is fragmented as well. Systems are holding data hostage. I have mentioned this problem before in a SWIB presentation. The issue was also recently touched upon in an OCLC Hanging Together blog post: "Synchronizing metadata among different databases" .

http://commonplace.net/2014/11/library-data-flows/

After this blog post about how to figure out the best ways to do data flow inventory, there were two more institutions that told us they were doing a similar thing.



Other Libraries

- hbz
  - Dataflows

Jörg Prante @xbib · Nov 27

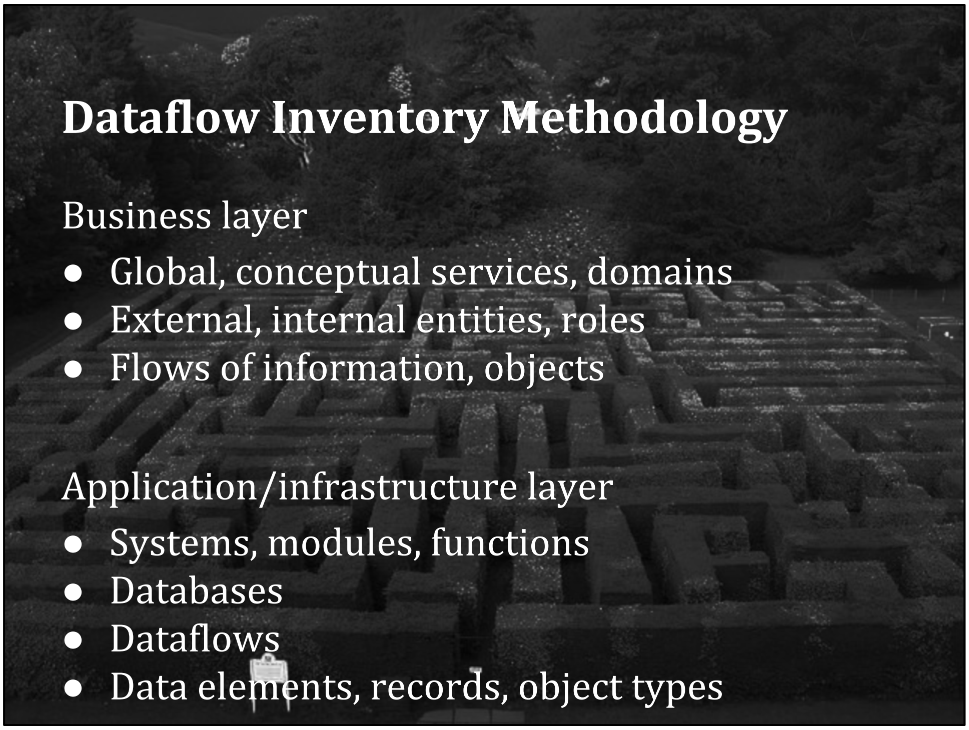
@lukask very interesting, since last week we at **hbz** also sketch **dataflow** processing diagrams for #Elasticsearch based catalog reconciliation

- Royal Library of The Netherlands
  - Metadata registry

The German regional Library Service Center hbz said they were doing similar things. We have not been in contact with hbz so far.

We have had talks and exchanges with the Royal Library of The Netherlands. They started from the other way, describing the data in a “metadata registry” using an Access database, whereas we started with describing flows and moving towards describing data.

Fortunately the Royal Library seemed to use the same perspectives and functional views :-)



Dataflow Inventory Methodology

Business layer

- Global, conceptual services, domains
- External, internal entities, roles
- Flows of information, objects

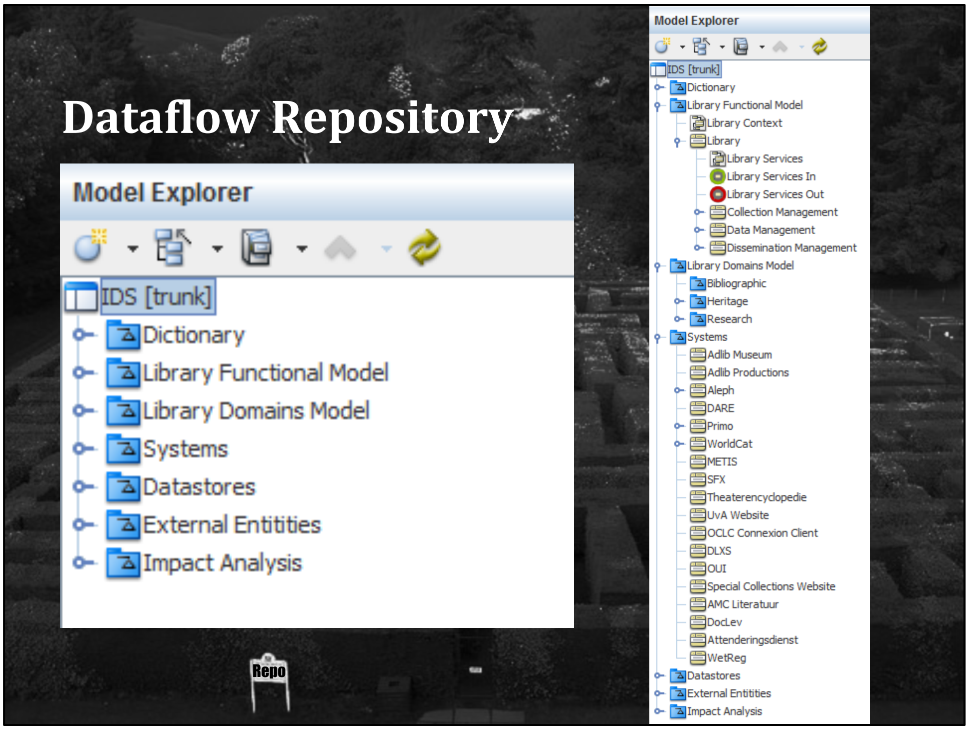
Application/infrastructure layer

- Systems, modules, functions
- Databases
- Dataflows
- Data elements, records, object types

The first months were needed for finding out how we were going to work, how to adapt DFD, how to work with Visual Paradigm, etc. By trial and error. In the end we came up with our “Methodology”, which is partly our adaptation of DFD, partly our adaptation of the tool, partly our inventory/repository structure.

We have a global distinction between Business Layer and Application+Infrastructure layer.
All described using DFD elements + Data Dictionary.
We distinguish certain “stereotypes” (a technique borrowed from UML, but possible in Visual Paradigm for DFD too) for “Processes” to identify “Domains”, “Services”, “Systems/Applications”, “Modules”, “Functions”etc.

Dataflow Repository



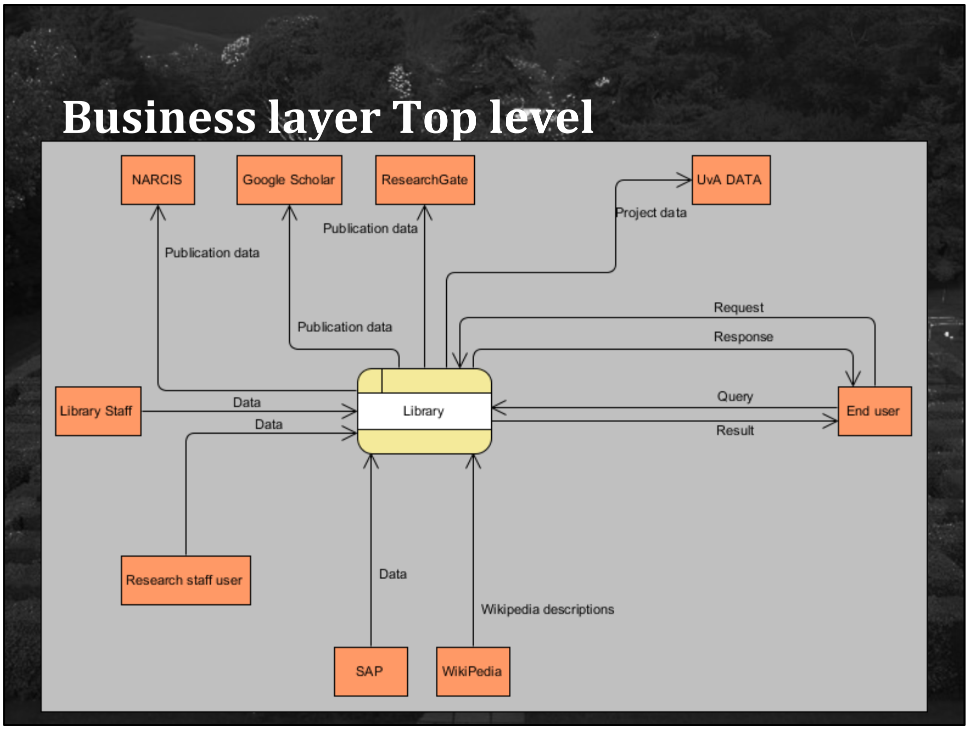
Model Explorer

- IDS [trunk]
  - Dictionary
  - Library Functional Model
  - Library Domains Model
  - Systems
  - Datastores
  - External Entitities
  - Impact Analysis

Model Explorer

- IDS [trunk]
  - Dictionary
  - Library Functional Model
    - Library Context
    - Library
      - Library Services
      - Library Services In
      - Library Services Out
      - Collection Management
      - Data Management
      - Dissemination Management
  - Library Domains Model
    - Bibliographic
    - Heritage
    - Research
  - Systems
    - Adlib Museum
    - Adlib Productions
    - Aleph
    - DARE
    - Primo
    - WorldCat
    - METIS
    - SFX
    - Theaterencyclopedie
    - UvA Website
    - OCLC Connexion Client
    - DLXS
    - OUI
    - Special Collections Website
    - AMC Literatuur
    - DocLev
    - Attenderingsdienst
    - WetReg
  - Datastores
  - External Entitities
  - Impact Analysis

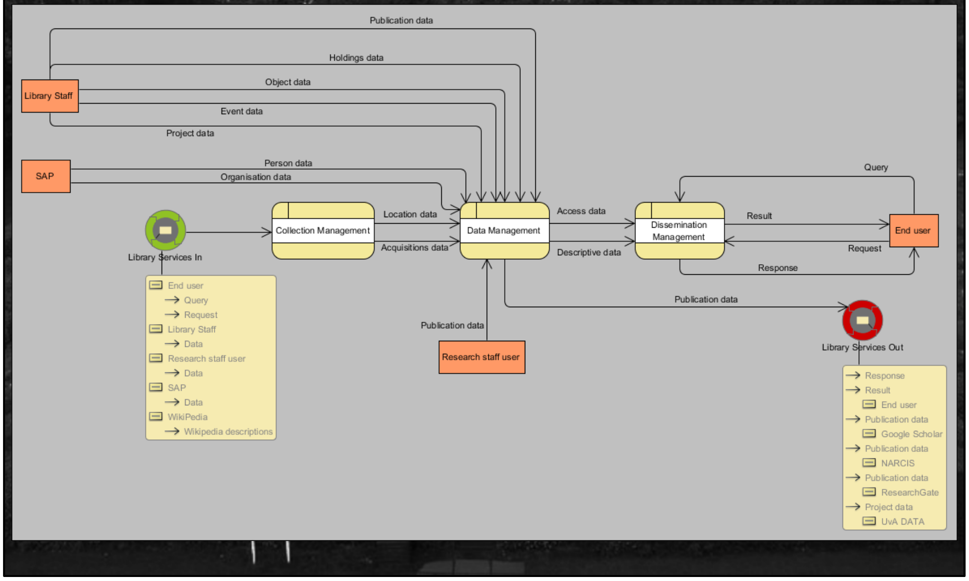
Our Visual Paradigm Project Model is subdivided into meaningful folders/submodels. In Visual Paradigm these submodels can also be used to generate focused reports.



Business layer Top level

Business Layer top level, incomplete example without all existing external entities.
In DFD: the top level “Context Model” has dataflows with all possible External Entities, that can be external systems, internal/external persons/roles, etc.
In our adaptation it is easier to use external entities also in lower levels.

Business layer Level 2

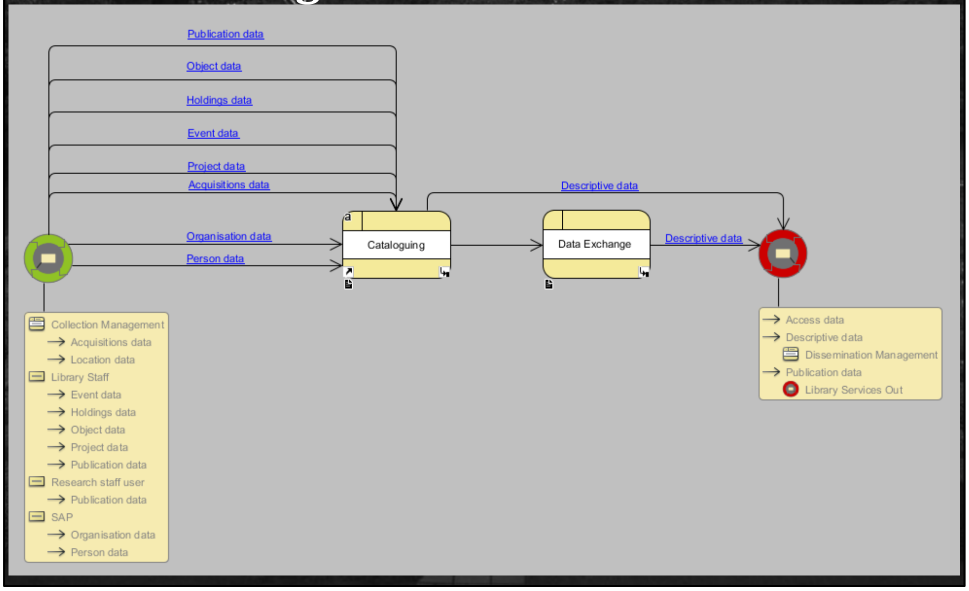


Business Layer second level.
Drilling down ("decomposition") into lower levels with decomposed processes.
Officially External Entities should only be drawn on the top level Context Model. Every flow that goes in or out must also be dealt with at the lower level

This level shows the main three Functional areas that we distinguish: "Collection Management" (the back office real world business) on one end, "Dissemination Management" (the front office business) on the other end, and in between the important "Data Management" Hub where connections are made between collection and end user services (discovery and delivery).

It is hard to draw solid lines between the three areas.

Business layer Level 3
Data Management

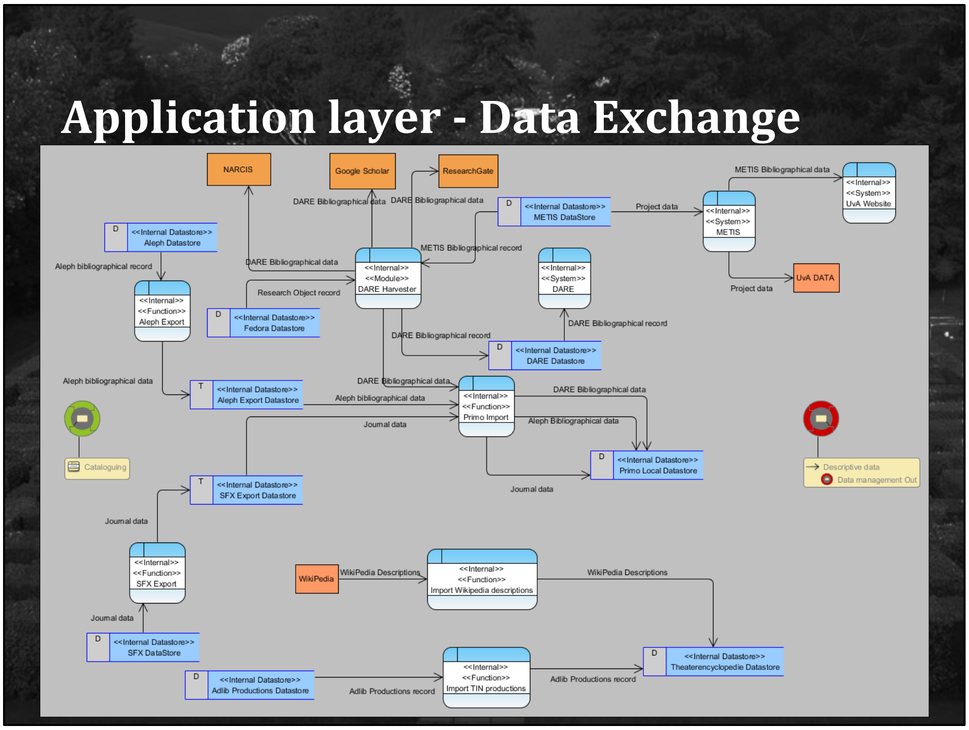


Publication data
Object data
Holdings data
Event data
Project data
Acquisitions data
Descriptive data
Organisation data
Person data
Cataloguing
Data Exchange
Descriptive data

Collection Management
→ Acquisitions data
→ Location data
Library Staff
→ Event data
→ Holdings data
→ Object data
→ Project data
→ Publication data
Research staff user
→ Publication data
SAP
→ Organisation data
→ Person data

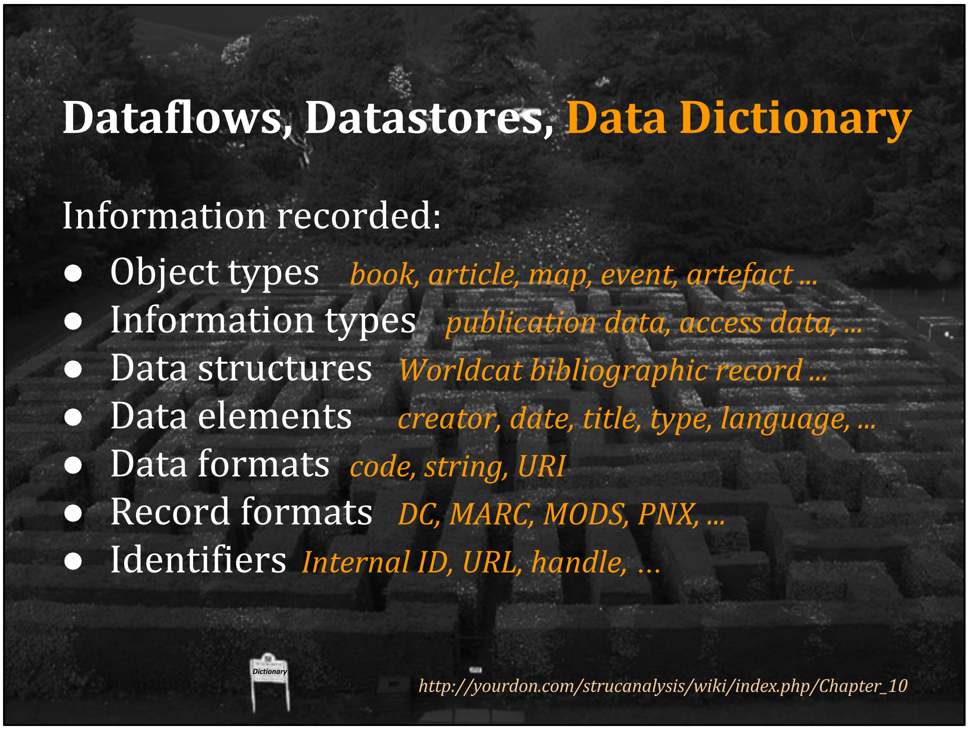
→ Access data
→ Descriptive data
Dissemination Management
→ Publication data
Library Services Out

Business Layer third level.
Drilling down Data Management: two areas of data:
Cataloguing (creation of data)
Data Exchange (moving data from cataloguing to dissemination: discovery and delivery services/datastores)



Application layer - Data Exchange

Application/Technology Infrastructure Layer.
Lowest level decomposition, showing dataflows between applications and databases. In this case in the Data Exchange area. This is just an incomplete snapshot. Typically we see here modules, scripts, functions that are part of a system or stand-alone, managing import and export of data between systems.



Dictionary

Dataflows, Datastores, Data Dictionary

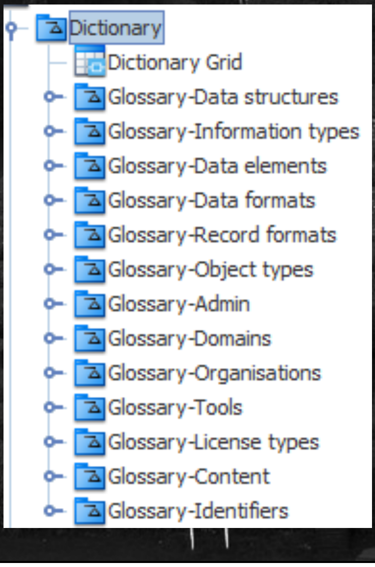
Information recorded:

- Object types *book, article, map, event, artefact ...*
- Information types *publication data, access data, ...*
- Data structures *Worldcat bibliographic record ...*
- Data elements *creator, date, title, type, language, ...*
- Data formats *code, string, URI*
- Record formats *DC, MARC, MODS, PNX, ...*
- Identifiers *Internal ID, URL, handle, ...*

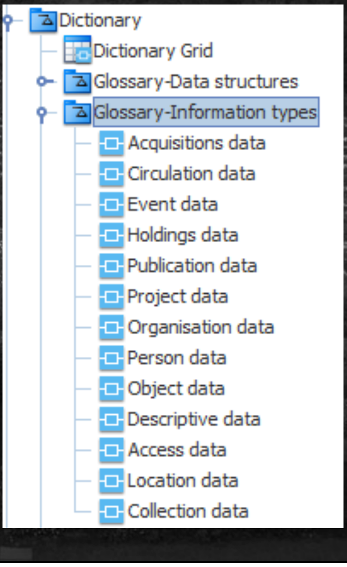
http://yourdon.com/strucanalysis/wiki/index.php/Chapter_10

For Dataflows and Datastores detailed information is recorded in the Data Dictionary. This is completely customizable.

Data dictionary



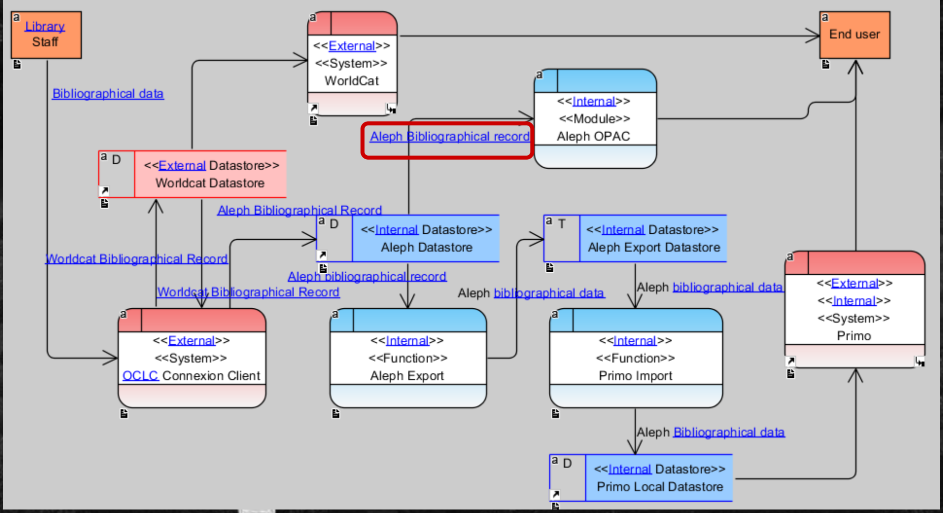
- Dictionary
  - Dictionary Grid
  - Glossary-Data structures
  - Glossary-Information types
  - Glossary-Data elements
  - Glossary-Data formats
  - Glossary-Record formats
  - Glossary-Object types
  - Glossary-Admin
  - Glossary-Domains
  - Glossary-Organisations
  - Glossary-Tools
  - Glossary-License types
  - Glossary-Content
  - Glossary-Identifiers



- Dictionary
  - Dictionary Grid
  - Glossary-Data structures
  - Glossary-Information types
    - Acquisitions data
    - Circulation data
    - Event data
    - Holdings data
    - Publication data
    - Project data
    - Organisation data
    - Person data
    - Object data
    - Descriptive data
    - Access data
    - Location data
    - Collection data

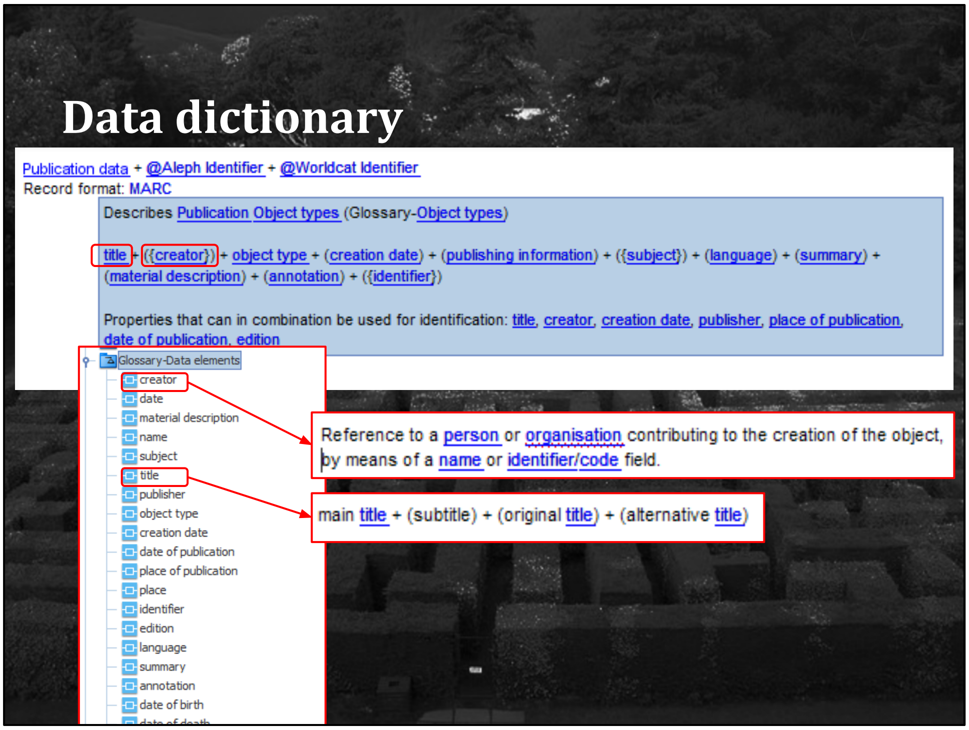
The Data Dictionary subfolder is divided into meaningful subfolders.
The data Dictionary Terms can be composed of multiple other Terms.

Data dictionary



All terms used in the models to describe Dataflows and other elements are automatically linked to terms in the Data Dictionary.

Data dictionary



Publication data + @Aleph Identifier + @Worldcat Identifier
Record format: MARC

Describes Publication Object types (Glossary-Object types)

title + ({creator}) + object type + (creation date) + (publishing information) + ({subject}) + (language) + (summary) + (material description) + (annotation) + ({identifier})

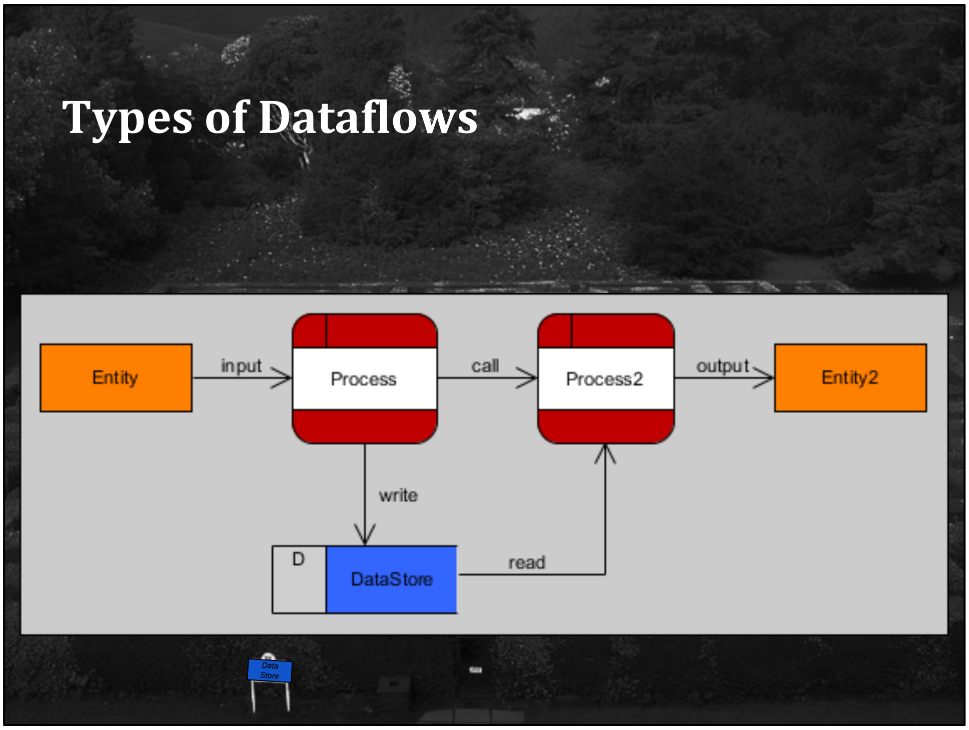
Properties that can in combination be used for identification: title, creator, creation date, publisher, place of publication, date of publication, edition

- Glossary-Data elements
  - creator
  - date
  - material description
  - name
  - subject
  - title
  - publisher
  - object type
  - creation date
  - date of publication
  - place of publication
  - place
  - identifier
  - edition
  - language
  - summary
  - annotation
  - date of birth
  - date of death

Reference to a person or organisation contributing to the creation of the object, by means of a name or identifier/code field.

main title + (subtitle) + (original title) + (alternative title)

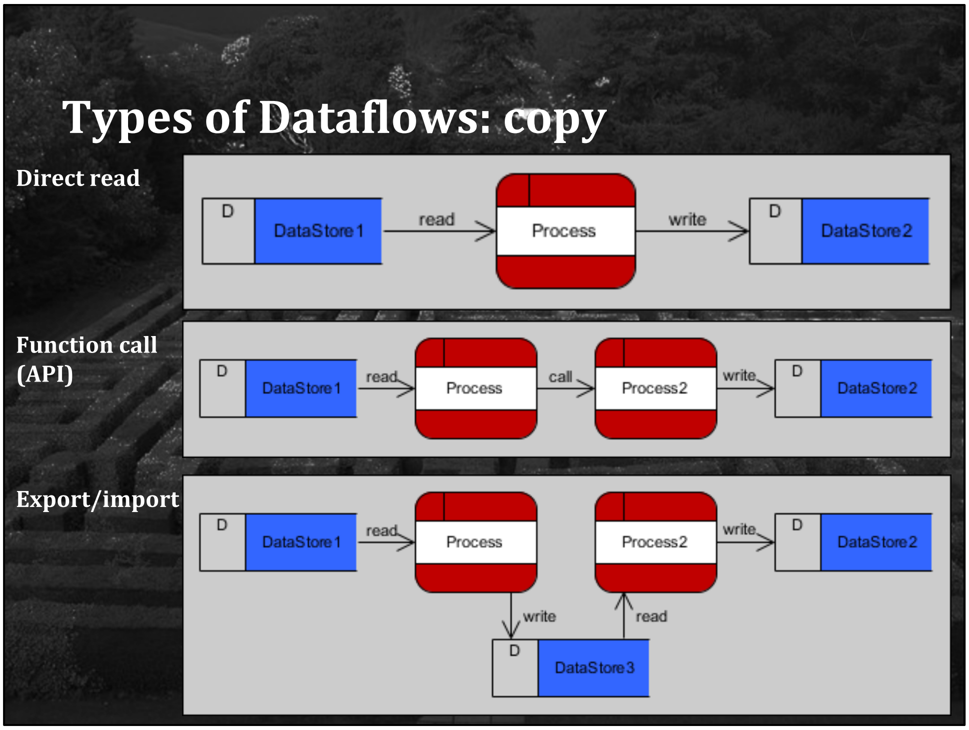
This is the Data Dictionar entry for “Aleph Bibliographical Record”.
Also all terms used in Data Dictionary descriptions are also linked to their own descriptions.
The official syntax used comprises composition, optionality, iteration, selection and aliases.



Types of Dataflows

We can distinguish 5 types of dataflows:

- A dataflow from an external entity representing an internal or external role, department, person, etc. to a process describes manual or automated data input into that process (system, module or function) and possibly, via that process, to a datastore.
- A dataflow from a datastore to a process represents the reading of data from a database or file by the process.
- A dataflow from a process to another process represents an exchange of data between systems, modules or functions by means of harvesting, API calls, push or similar.
- A dataflow from a process to an external entity represents output to an end user interface (screen, export, report, etc.) or external system.
- A dataflow from a process to a datastore indicates the writing of data to a database or file.



Types of Dataflows: copy

Direct read

Function call (API)

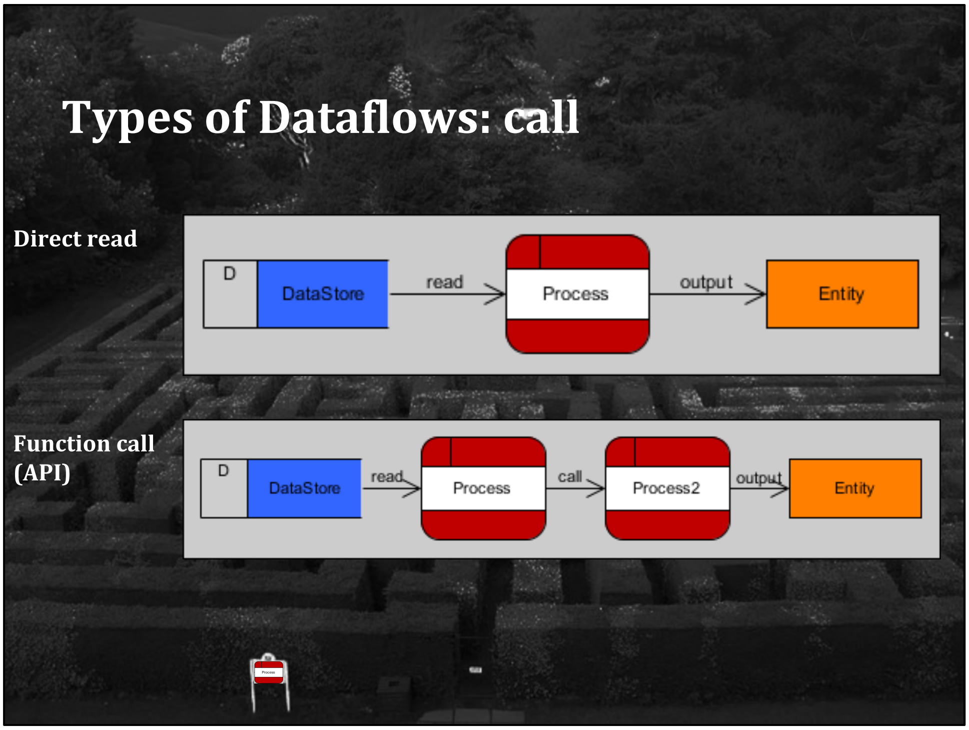
Export/import

Dataflows that copy data between datastores (start and endpoint flows "read" and "write") have three forms:

Direct read: A system, module or function reads directly from the source datastore and writes directly to the destination datastore (Direct reading of a database by an external system is hardly ever possible; this would be the scenario for real Linked Open Data. Direct writing to a database by an external system is hardly ever possible either).

Function call: A system, module or function sends a call to another system, module or function, requesting information, receives data from that system, module or function, and writes to the destination datastore (the call between two systems is usually done via the dedicated API of the called system; another implementation is harvesting via OAI)

Export/import: A system, module or function reads directly from the source datastore and writes to a temporary file; another system, module or function reads from the temporary file and writes to the destination datastore

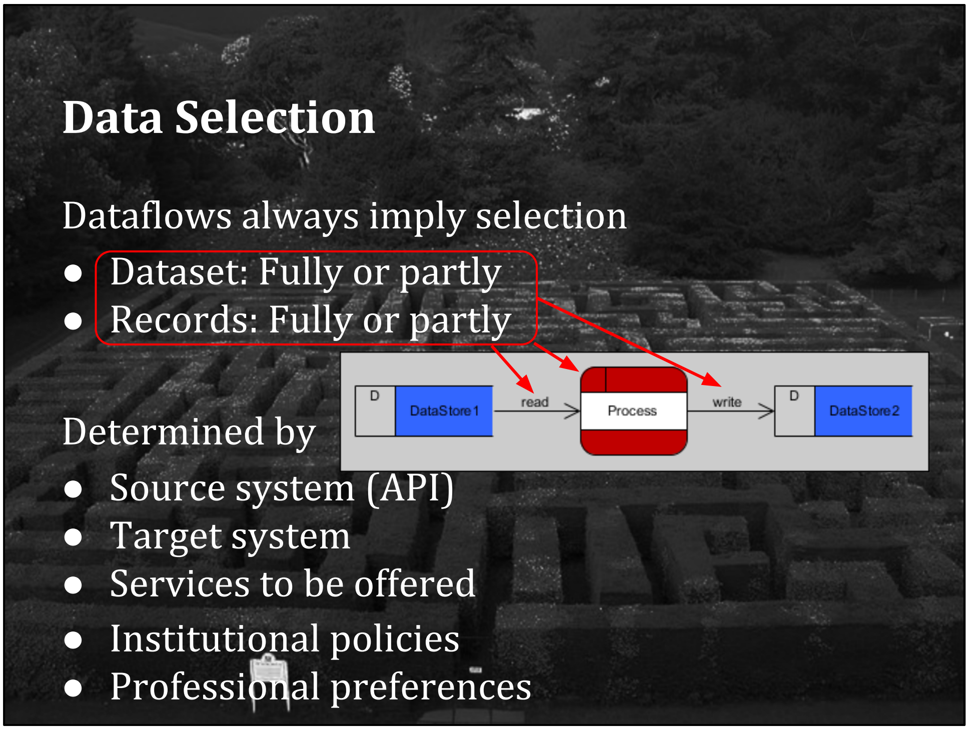


Types of Dataflows: call

Direct read

Function call (API)

Dataflows that use data from datastores for direct output have two forms:
Direct read: A system, module or function reads directly from the source datastore and sends directly to the destination entity; this is typically the way a single system works through its own user interface (Again, direct reading of a database by an external system is hardly ever possible)
Function call: A system, module or function sends a call to another system, module or function, requesting information, receives data from that system, module or function, and sends to the destination entity (again here the call is done using the dedicated API of the called system)



Data Selection

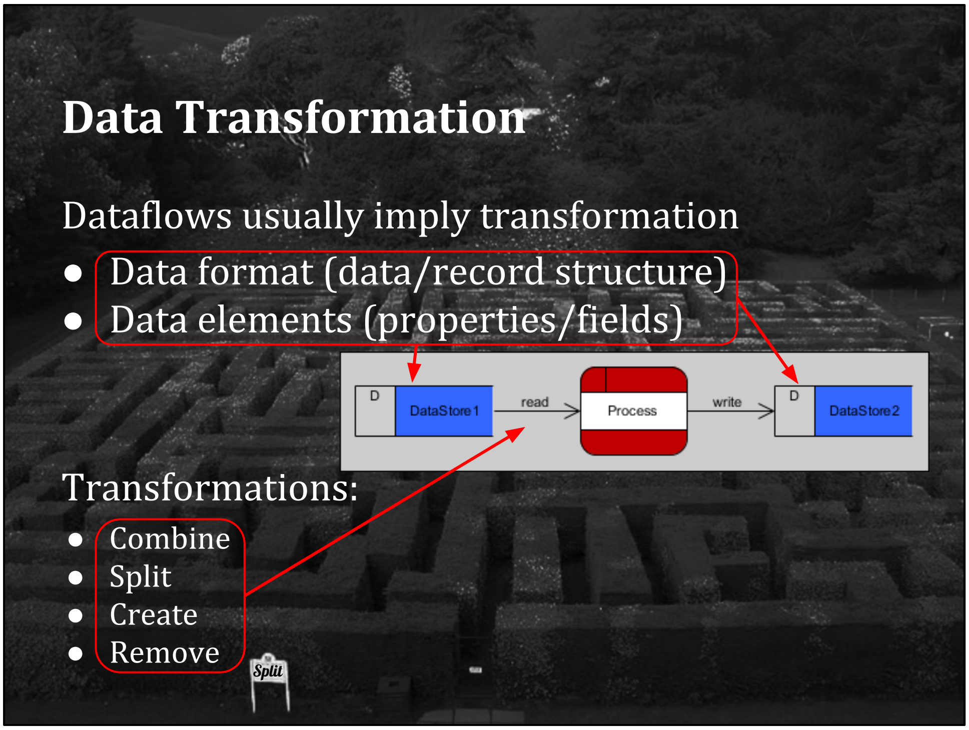
Dataflows always imply selection

- Dataset: Fully or partly
- Records: Fully or partly

Determined by

- Source system (API)
- Target system
- Services to be offered
- Institutional policies
- Professional preferences

Dataflows always imply selection, by systems, institutional policies, professional preferences: whole or part of datasource, which data elements?
Selection must be documented in order to be transparent.



Data Transformation

Dataflows usually imply transformation

- Data format (data/record structure)
- Data elements (properties/fields)

Transformations:

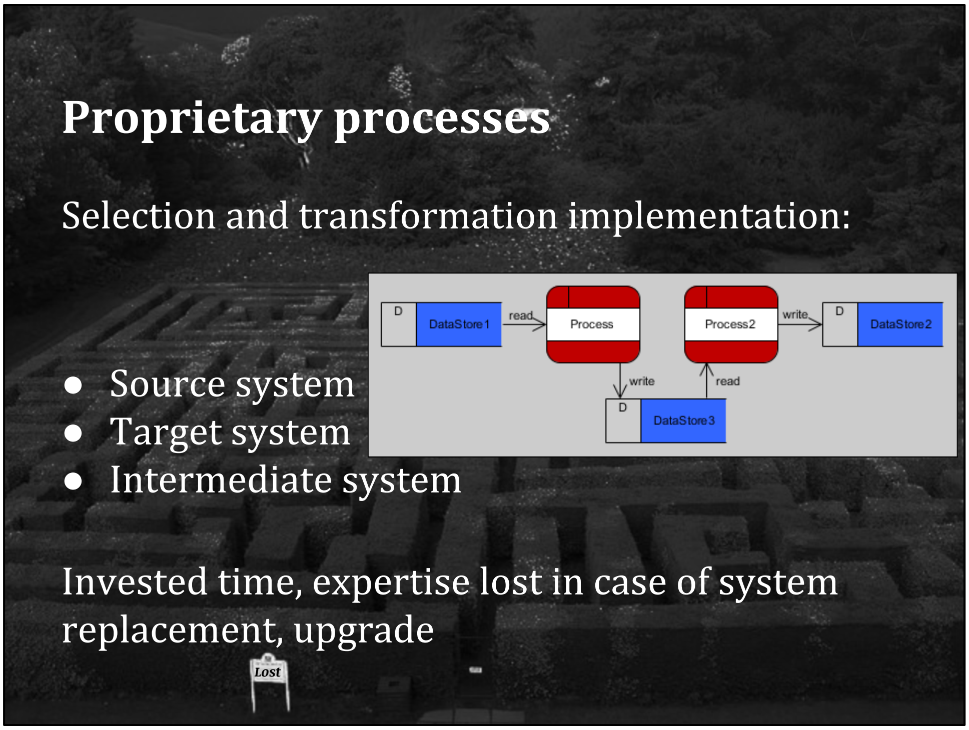
- Combine
- Split
- Create
- Remove

Dataflows between systems/datastores imply transformation of data from one format to another.

Transformation can be done on two levels: format (data structure) and fields (data elements).

Some data elements have to be combined, some have to be split up, some have to be left out, some have to be created.

Transformation must be documented in order to be transparent.



Proprietary processes

Selection and transformation implementation:

- Source system
- Target system
- Intermediate system

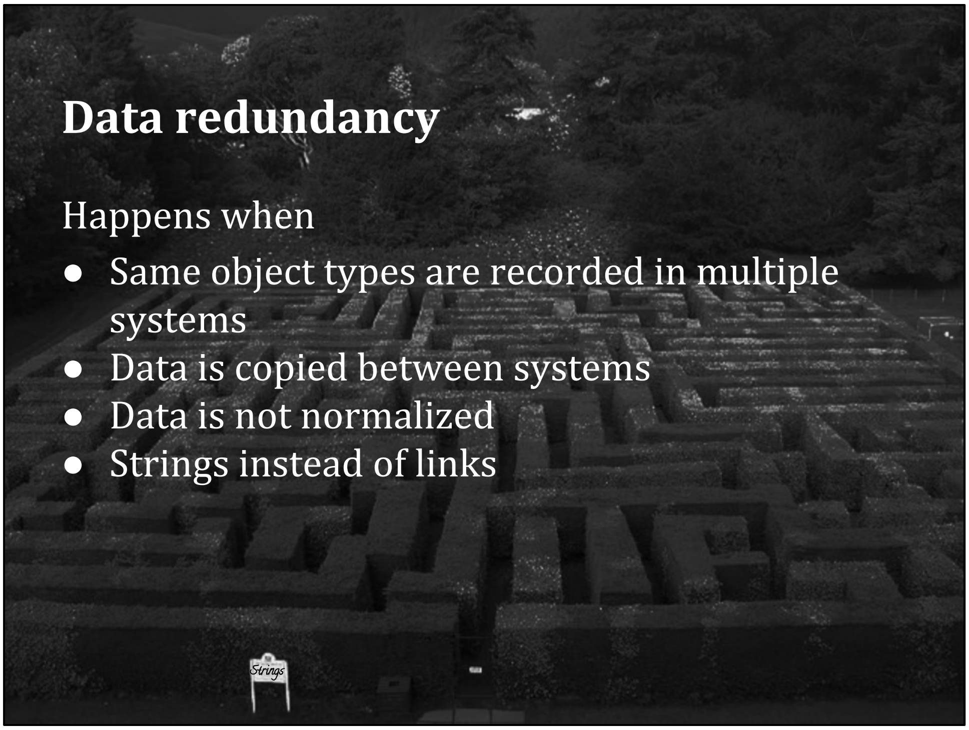
Invested time, expertise lost in case of system replacement, upgrade

D
DataStore1
read
Process
Process2
write
D
DataStore2
write
read
D
DataStore3
Lost

In many cases input and output dataflows, with selection and transformation, are performed by proprietary functions and configurations, belonging to source and/or target systems.

This constitutes an additional complication in administering the complete systems and data infrastructure, if and when a system is replaced and not only data, but also selection and transformation procedures have to be converted. A considerable investment and a substantial body of knowledge used for essential data transformations become useless and have to be created from scratch.

This must be documented.

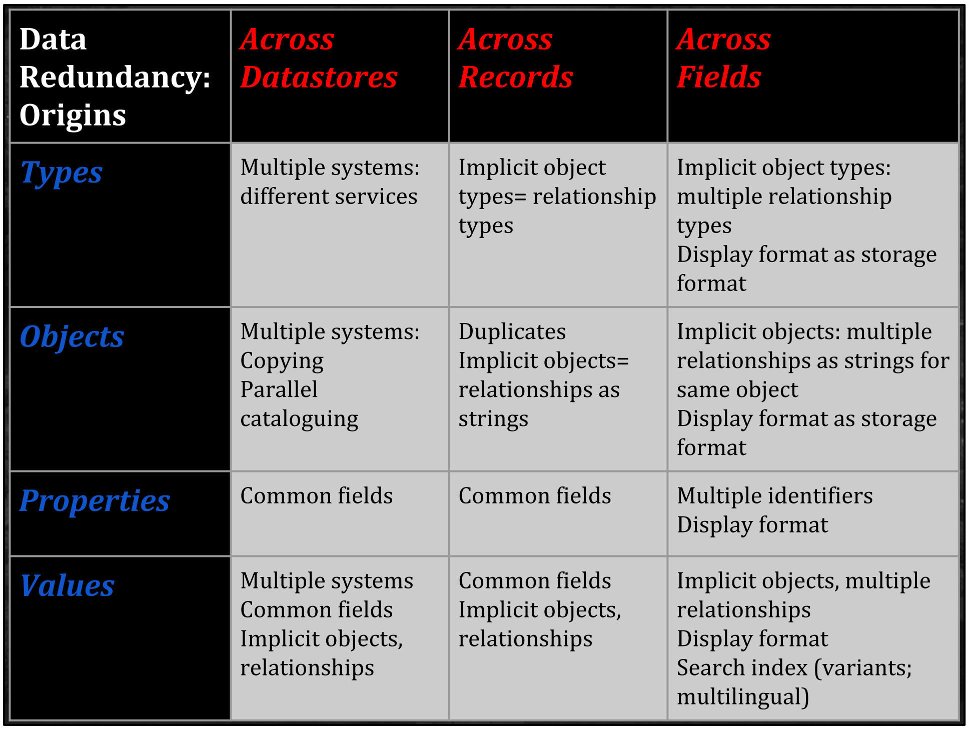


Data redundancy

Happens when

- Same object types are recorded in multiple systems
- Data is copied between systems
- Data is not normalized
- Strings instead of links

Data redundancy is the core issue in data infrastructure management.

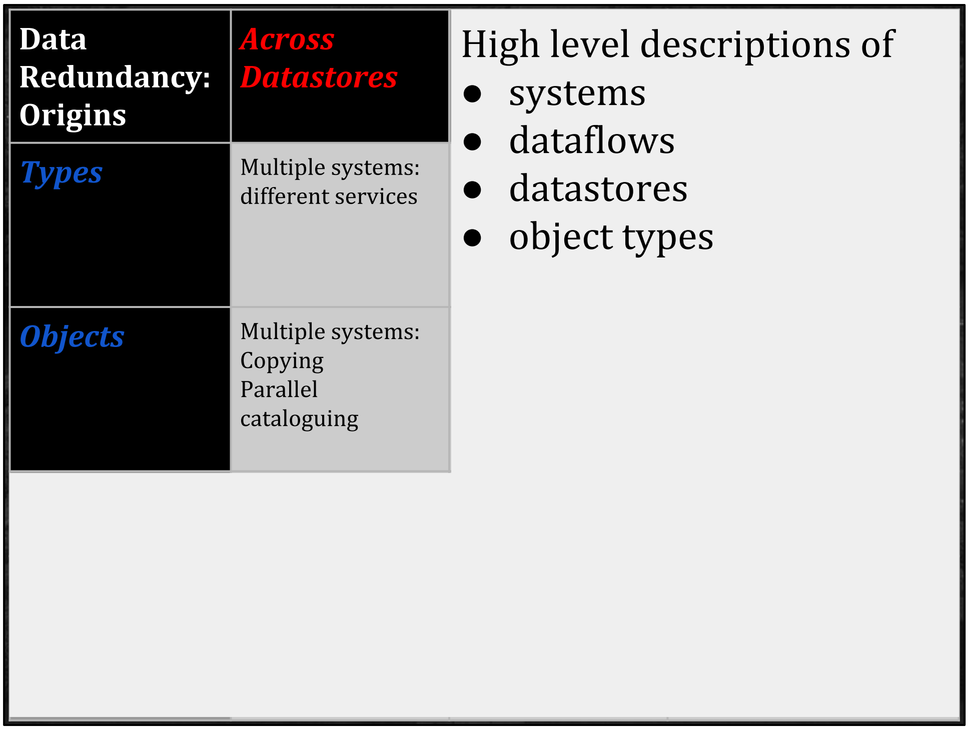


Data Redundancy: Origins	*Across Datastores*	*Across Records*	*Across Fields*
Types	Multiple systems: different services	Implicit object types= relationship types	Implicit object types: multiple relationship types Display format as storage format
Objects	Multiple systems: Copying Parallel cataloguing	Duplicates Implicit objects= relationships as strings	Implicit objects: multiple relationships as strings for same object Display format as storage format
Properties	Common fields	Common fields	Multiple identifiers Display format
Values	Multiple systems Common fields Implicit objects, relationships	Common fields Implicit objects, relationships	Implicit objects, multiple relationships Display format Search index (variants; multilingual)

Data redundancy typology. The origins of redundancy.
Levels and scopes matrix.

- Type redundancy: Certain object/content types (such as books, images, people, articles, artefacts) can be managed with multiple systems/datastores (for instance "books" in a CRIS and an ILS).
  - Type redundancy is also possible within a system/datastore: in multiple records (in that case we see implicit object types=relationships as strings, such as "persons" as "creator", "subject", etc.)
  - Type redundancy within one record: for instance MARC "author": tag 100 (person as creator), 700 (organization as creator), and subfield 245 $c (Title: Statement of responsibility - person or organization as creator)
- Object redundancy implies Type redundancy: actual items are recorded in multiple systems/datastores (for instance a dissertation in a CRIS, but also in the ILS), multiple records (for instance a specific person as creator, subject in multiple records), multiple fields (for instance the same person as creator and subject: an autobiography!).
- Property redundancy: the same fields (common fields such as Title, Creator, Date, Subject) are managed in multiple systems/datastores, records, fields. Special case: unique identifiers in one record can consist of multiple fields and combinations of fields: internal ID, ISBN, DOI, ORCID, VIAF id, or even Journal title+Volume+Issue+pages).

- Value redundancy: again these are implicit relationships as strings; and also variants (for instance multilingual) representing the same object, for search/retrieve purposes

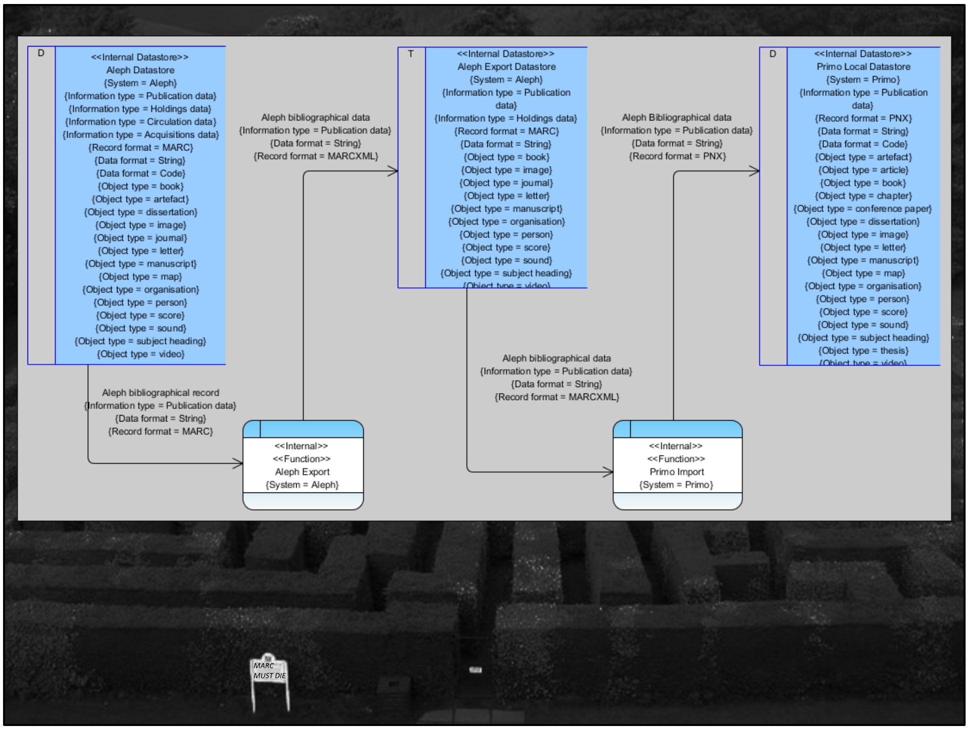


Data Redundancy: Origins	*Across Datastores*
Types	Multiple systems: different services
Objects	Multiple systems: Copying Parallel cataloguing

High level descriptions of

- systems
- dataflows
- datastores
- object types

Type and Object redundancy across datastores can be recorded and discovered in the tool with high level descriptions of Dataflows and Datastores.



D

<<Internal Datastore>>
Aleph Datastore
{System = Aleph}
{Information type = Publication data}
{Information type = Holdings data}
{Information type = Circulation data}
{Information type = Acquisitions data}
{Record format = MARC}
{Data format = String}
{Data format = Code}
{Object type = book}
{Object type = artefact}
{Object type = dissertation}
{Object type = image}
{Object type = journal}
{Object type = letter}
{Object type = manuscript}
{Object type = map}
{Object type = organisation}
{Object type = person}
{Object type = score}
{Object type = sound}
{Object type = subject heading}
{Object type = video}

Aleph bibliographical record
{Information type = Publication data}
{Data format = String}
{Record format = MARC}

<<Internal>>
<<Function>>
Aleph Export
{System = Aleph}

Aleph bibliographical data
{Information type = Publication data}
{Data format = String}
{Record format = MARCXML}

T

<<Internal Datastore>>
Aleph Export Datastore
{System = Aleph}
{Information type = Publication data}
{Information type = Holdings data}
{Record format = MARC}
{Data format = String}
{Object type = book}
{Object type = image}
{Object type = journal}
{Object type = letter}
{Object type = manuscript}
{Object type = organisation}
{Object type = person}
{Object type = score}
{Object type = sound}
{Object type = subject heading}
{Object type = video}

Aleph bibliographical data
{Information type = Publication data}
{Data format = String}
{Record format = MARCXML}

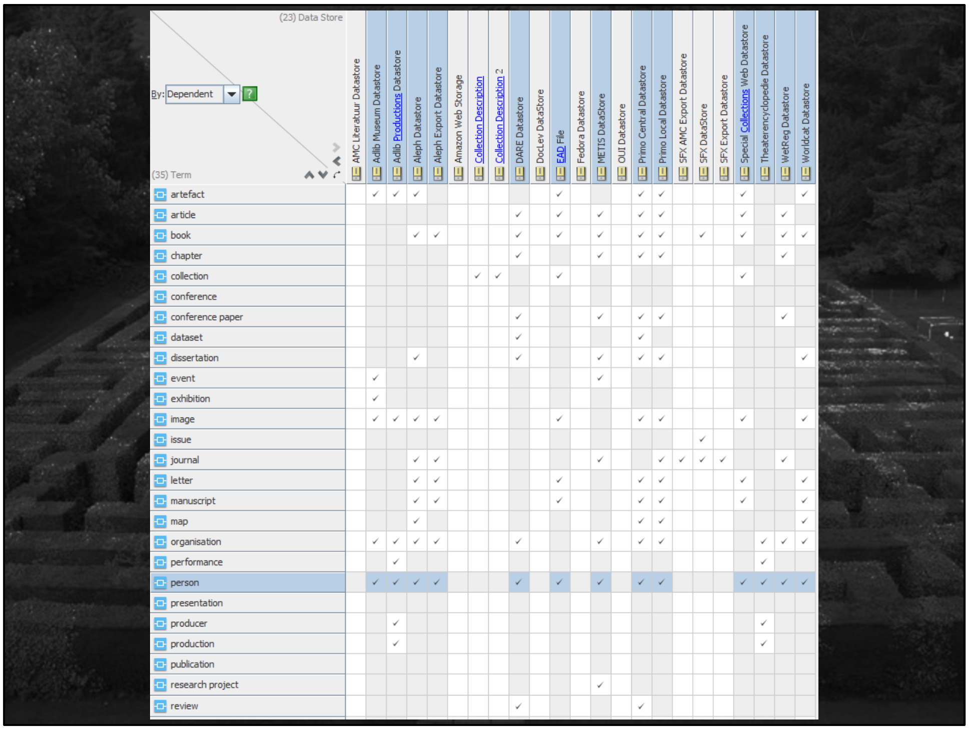
<<Internal>>
<<Function>>
Primo Import
{System = Primo}

Aleph Bibliographical data
{Information type = Publication data}
{Data format = String}
{Record format = PNX}

D

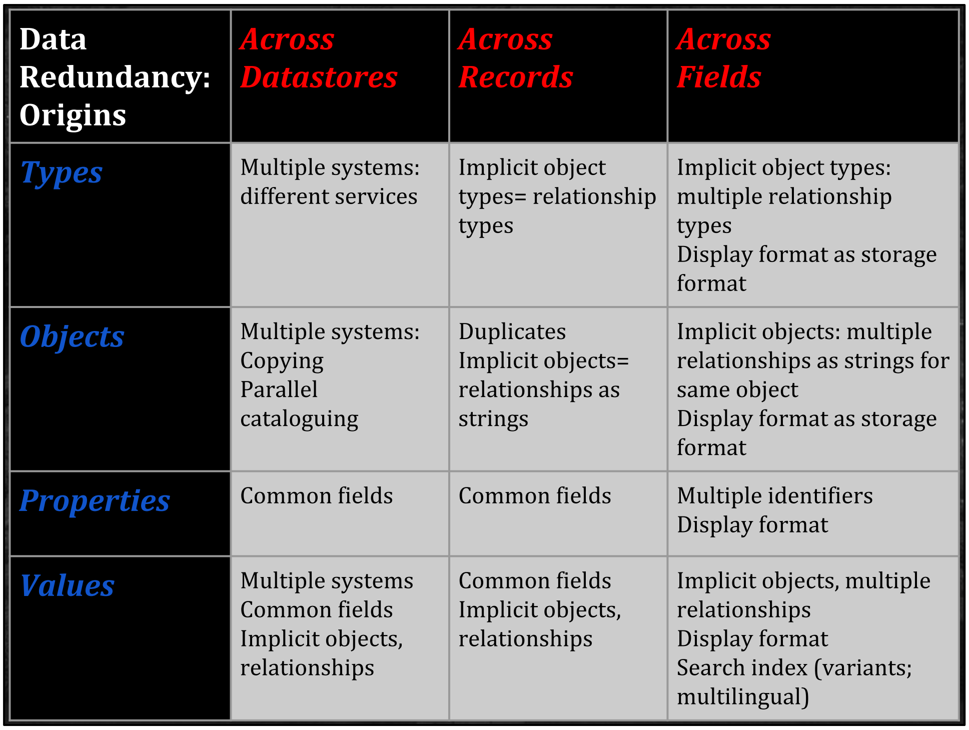
<<Internal Datastore>>
Primo Local Datastore
{System = Primo}
{Information type = Publication data}
{Record format = PNX}
{Data format = String}
{Data format = Code}
{Object type = artefact}
{Object type = article}
{Object type = book}
{Object type = chapter}
{Object type = conference paper}
{Object type = dissertation}
{Object type = image}
{Object type = letter}
{Object type = manuscript}
{Object type = map}
{Object type = organisation}
{Object type = person}
{Object type = score}
{Object type = sound}
{Object type = subject heading}
{Object type = thesis}
{Object type = video}

This is an example of object types per datastore and dataflow in a situation of copying data between backend (Collection Management) and frontend (Dissemination Management) systems.



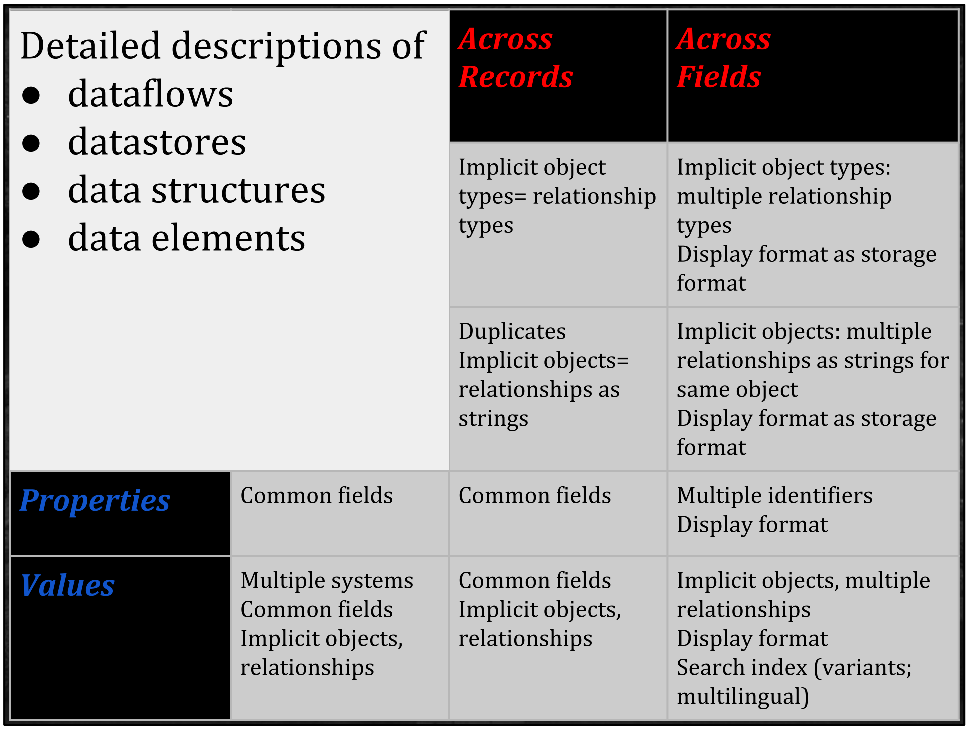
(35) Term	AMC Literatuur Datastore	Adlib Museum Datastore	Adlib Productions Datastore	Aleph Datastore	Aleph Export Datastore	Amazon Web Storage	Collection Description	Collection Description 2	DARE Datastore	DocLev DataStore	EAD File	Fedora Datastore	METIS DataStore	OUI Datastore	Primo Central Datastore	Primo Local Datastore	SFX AMC Export Datastore	SFX DataStore	SFX Export Datastore	Special Collections Web Datastore	Theaterencyclopedie Datastore	WetReg Datastore	Worldcat Datastore
artefact		✓	✓	✓							✓				✓	✓				✓			✓
article									✓		✓		✓		✓	✓				✓		✓	
book				✓	✓				✓		✓		✓		✓	✓		✓		✓		✓	✓
chapter									✓				✓		✓	✓						✓	
collection							✓	✓			✓									✓			
conference																							
conference paper									✓				✓		✓	✓						✓	
dataset									✓						✓								
dissertation				✓					✓				✓		✓	✓							✓
event		✓											✓										
exhibition		✓																					
image		✓	✓	✓	✓						✓				✓	✓				✓			✓
issue																		✓					
journal				✓	✓								✓			✓	✓	✓	✓			✓	
letter				✓	✓						✓				✓	✓				✓			✓
manuscript				✓	✓						✓				✓	✓				✓			✓
map				✓											✓	✓							✓
organisation		✓	✓	✓	✓				✓				✓		✓	✓					✓	✓	✓
performance			✓																		✓		
person		✓	✓	✓	✓				✓		✓		✓		✓	✓				✓	✓	✓	✓
presentation																							
producer			✓																		✓		
production			✓																		✓		
publication																							
research project													✓										
review									✓						✓								

Visual Paradigm provides automatic matrix overviews of relations between datastores and object types, if recorded correctly.



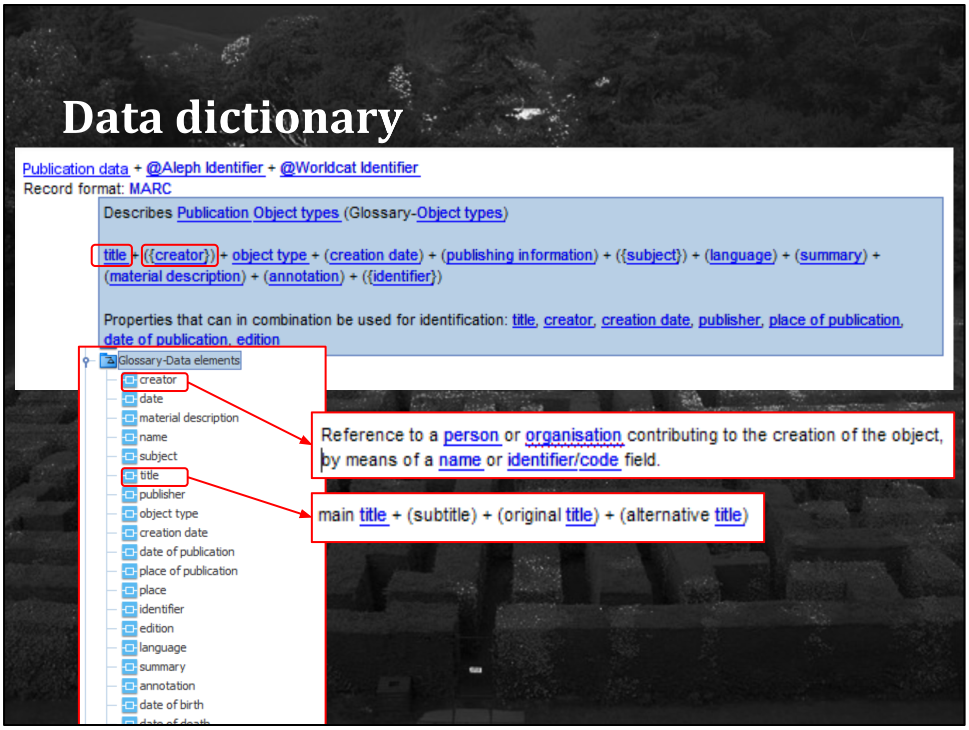
Data Redundancy: Origins	*Across Datastores*	*Across Records*	*Across Fields*
Types	Multiple systems: different services	Implicit object types= relationship types	Implicit object types: multiple relationship types Display format as storage format
Objects	Multiple systems: Copying Parallel cataloguing	Duplicates Implicit objects= relationships as strings	Implicit objects: multiple relationships as strings for same object Display format as storage format
Properties	Common fields	Common fields	Multiple identifiers Display format
Values	Multiple systems Common fields Implicit objects, relationships	Common fields Implicit objects, relationships	Implicit objects, multiple relationships Display format Search index (variants; multilingual)

Back to the data redundancy typology matrix.



Detailed descriptions of ● dataflows ● datastores ● data structures ● data elements		***Across Records***	***Across Fields***
		Implicit object types= relationship types	Implicit object types: multiple relationship types Display format as storage format
		Duplicates Implicit objects= relationships as strings	Implicit objects: multiple relationships as strings for same object Display format as storage format
Properties	Common fields	Common fields	Multiple identifiers Display format
Values	Multiple systems Common fields Implicit objects, relationships	Common fields Implicit objects, relationships	Implicit objects, multiple relationships Display format Search index (variants; multilingual)

For data redundancy within systems/datastores, records, and for Property and Value redundancy we need more detailed descriptions, recorded in the Data Dictionary.



Data dictionary

Publication data + @Aleph Identifier + @Worldcat Identifier
Record format: MARC

Describes Publication Object types (Glossary-Object types)

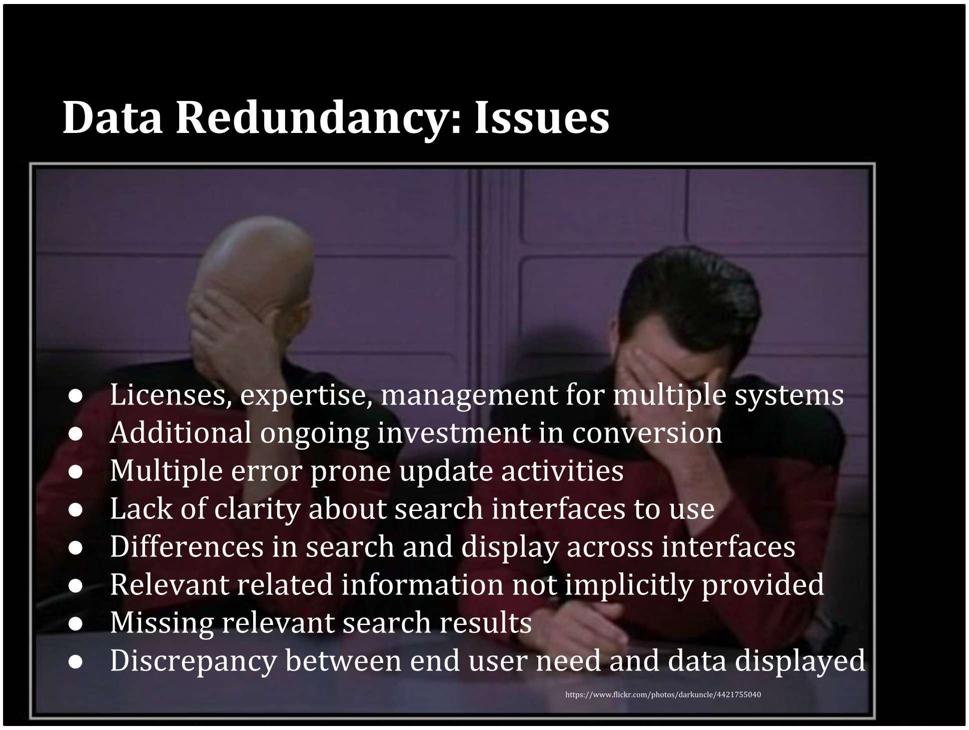
title + ({creator}) + object type + (creation date) + (publishing information) + ({subject}) + (language) + (summary) + (material description) + (annotation) + ({identifier})

Properties that can in combination be used for identification: title, creator, creation date, publisher, place of publication, date of publication, edition

Reference to a person or organisation contributing to the creation of the object, by means of a name or identifier/code field.

main title + (subtitle) + (original title) + (alternative title)

Again the Data Dictionary example.

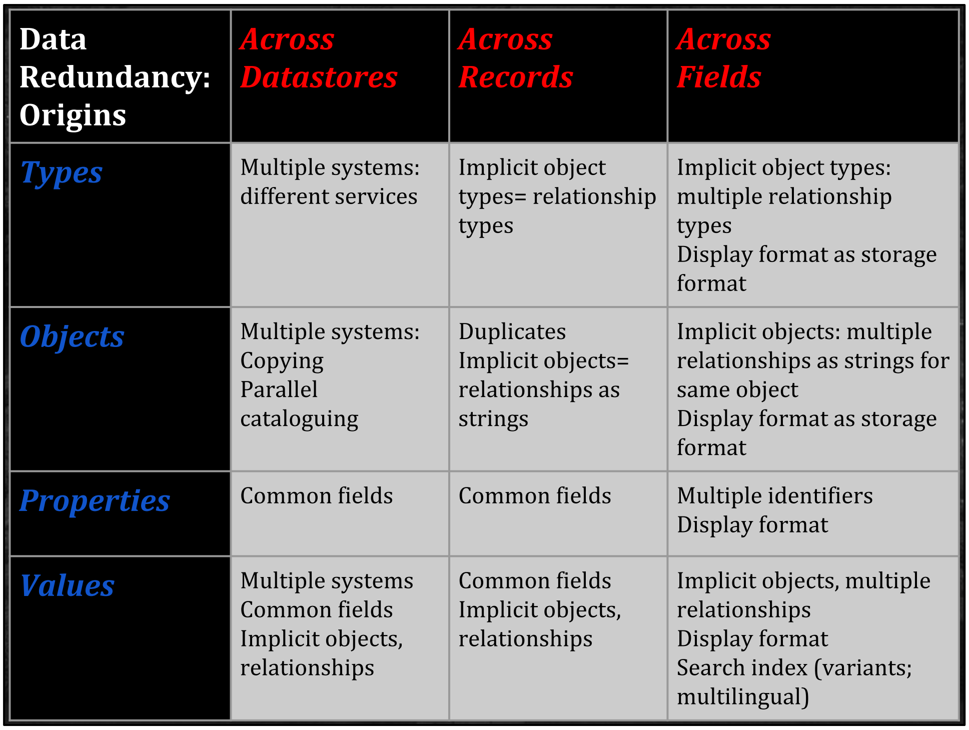


Data Redundancy: Issues

- Licenses, expertise, management for multiple systems
- Additional ongoing investment in conversion
- Multiple error prone update activities
- Lack of clarity about search interfaces to use
- Differences in search and display across interfaces
- Relevant related information not implicitly provided
- Missing relevant search results
- Discrepancy between end user need and data displayed

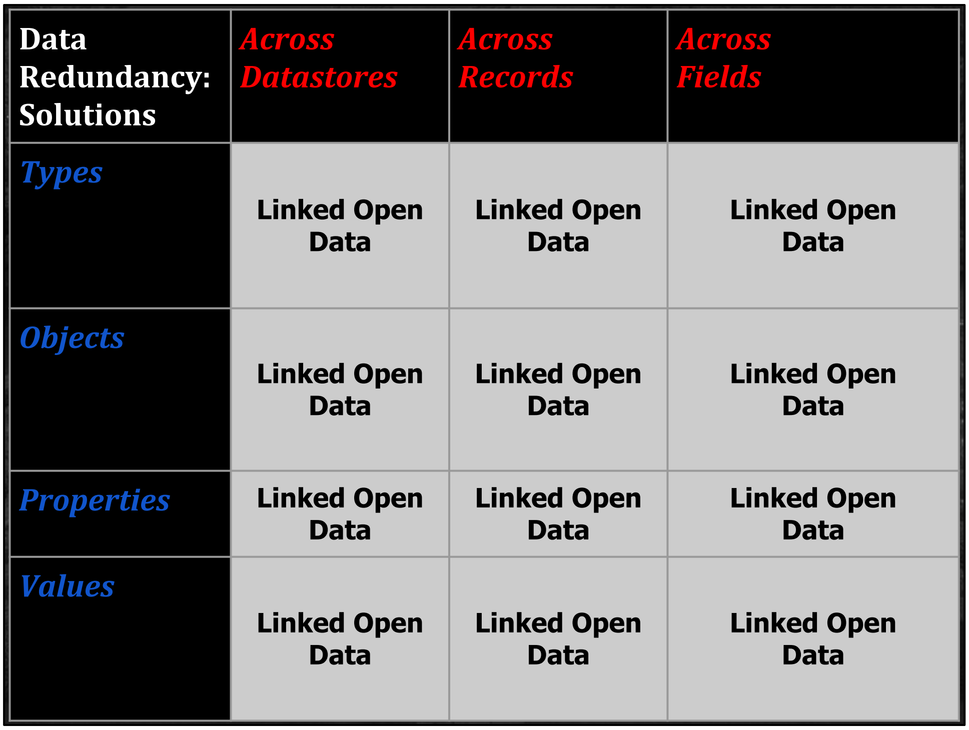
https://www.flickr.com/photos/darkuncle/4421755040

It is important to understand data redundancy because it is the possible cause of a substantial number of bottlenecks.



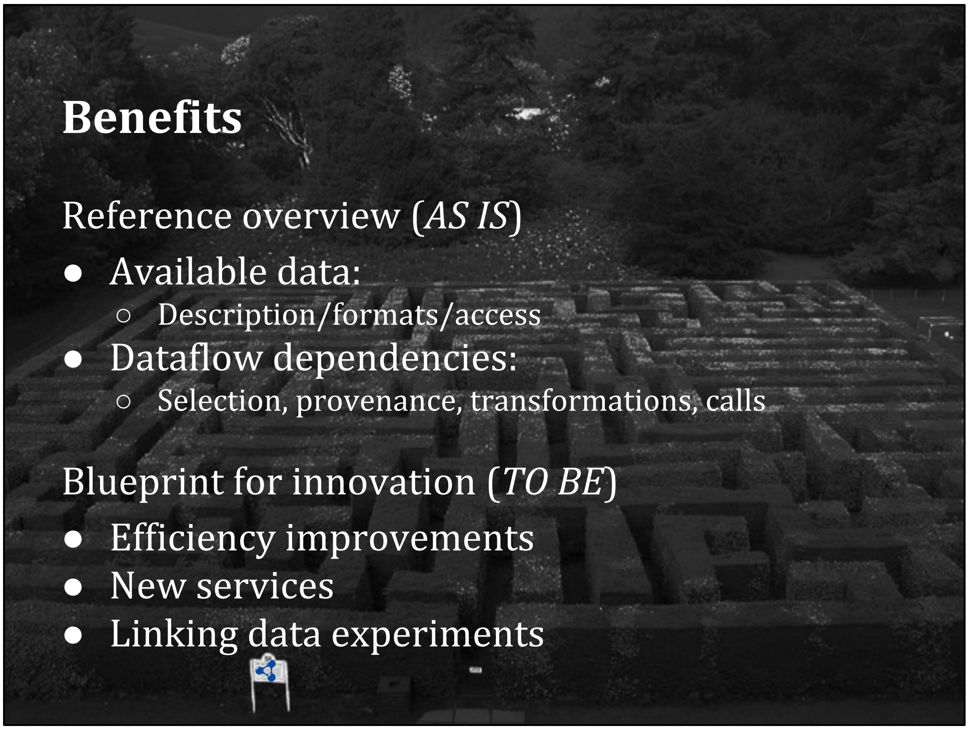
Data Redundancy: Origins	*Across Datastores*	*Across Records*	*Across Fields*
Types	Multiple systems: different services	Implicit object types= relationship types	Implicit object types: multiple relationship types Display format as storage format
Objects	Multiple systems: Copying Parallel cataloguing	Duplicates Implicit objects= relationships as strings	Implicit objects: multiple relationships as strings for same object Display format as storage format
Properties	Common fields	Common fields	Multiple identifiers Display format
Values	Multiple systems Common fields Implicit objects, relationships	Common fields Implicit objects, relationships	Implicit objects, multiple relationships Display format Search index (variants; multilingual)

A final look at the Data Redundancy Typology Matrix: the orgins.



Data Redundancy: Solutions	*Across Datastores*	*Across Records*	*Across Fields*
Types	**Linked Open Data**	**Linked Open Data**	**Linked Open Data**
Objects	**Linked Open Data**	**Linked Open Data**	**Linked Open Data**
Properties	**Linked Open Data**	**Linked Open Data**	**Linked Open Data**
Values	**Linked Open Data**	**Linked Open Data**	**Linked Open Data**

Solutions for all these issues would ideally by some form of linked open data.



Benefits

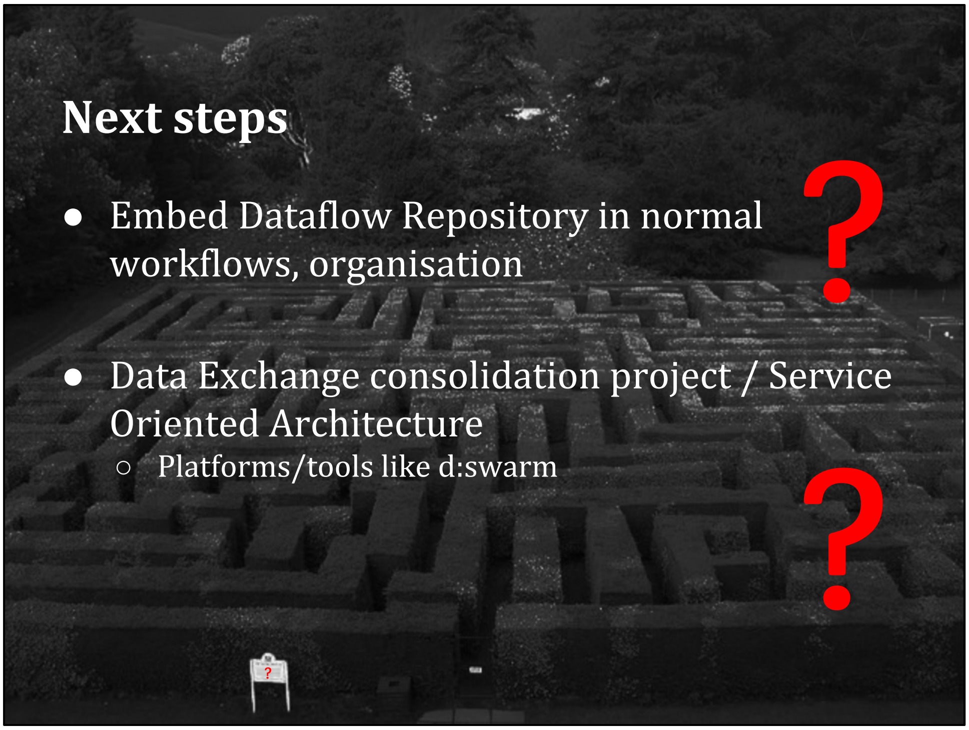
Reference overview (*AS IS*)

- Available data:
  - Description/formats/access
- Dataflow dependencies:
  - Selection, provenance, transformations, calls

Blueprint for innovation (*TO BE*)

- Efficiency improvements
- New services
- Linking data experiments

To conclude, the benefits of maintaining a living dataflow repository are to provide an overview of the available data, to help identify dependencies and bottlenecks in the library information infrastructure, and to offer a number of clues and starting points for the elimination of these bottlenecks and for the improvement of workflows and end user services



Next steps

- Embed Dataflow Repository in normal workflows, organisation
- Data Exchange consolidation project / Service Oriented Architecture
  - Platforms/tools like d:swarm

For this to be useful, dataflow repository management should be part of normal workflows and be managed in a structural way.

A feasible intermediate term Innovative Data Management/Exchange project could be: to develop an intermediate system independent data layer, like d:swarm.

Special thanks to Céline Carty

Special thanks to Céline Carty, without whom this Thank you slide and the following Final slide would not be here. ;-)

Background image http://commons.wikimedia.org/wiki/File%3ATraquair_House_Maze.jpg
http://outlawjimmy.com/2013/04/24/thats-all-folks/