CLIO-DAP:
System Analysis and Design

Leen Breure

in collaboration with
Maarten Hoogerwerf
Georgi Khomeriki

Universiteit Utrecht
departement Informatica

The Hague 2013 – version 1.2

Contents

Table of figures

1. Introduction

1.1. Conclusions from the orientation phase

On April 3 2013 the CLIO-INFRA stakeholders in this project discussed the report Data *Availability Policies: Ideal and Practice* (the deliverable of the first work package)[1], hereafter indicated as "the report", and decided on the following principles for the CLIO-DAP demonstrator:

1. We shall propose to the journals:
    a. a simple DAP, see report section 5.4.2, p. 32. Authors should submit data documentation sufficiently detailed for replication;
    b. a non-mandatory data review policy as outlined in report section 5.4.3, p. 32-34 (i.e. a separation between publication review and data review);
    c. to choose DANS as preferred repository (see report section 5.3, p. 30).

    Of course, journals remain free to decide otherwise.

2. For the time being we postpone the contact with foreign journals (through email and Skype, as proposed by Breure and Hoogerwerf). Instead, the involved editors will be informed in a meeting somewhere in 2013. During the next months we will first strive for more concrete results in close collaboration with the two Dutch journals, namely:
    a. *International Review of Social History* (Cambridge University Press)[2]
    b. *Tijdschrift voor Sociale en Economische Geschiedenis* (Amsterdam University Press)[3]

3. DANS will provide facilities for a simple, non-interactive data review paper, which includes a list of criteria for both authors and reviewers. Writing such a data review paper is an option, as described in section 5.4.3 of the report (p. 33).

1.2. Implications

These conclusions have some implications for the project implementation, compared with what has been proposed:

1. Ad 1.1: **journals**: *"Of these journals, we will record the following information: Publisher, Editor(s), Email of the editor, Language(s), Country of publisher, Website, Email of the journal, DAP (Y/N), Link to DAP, Editorial/Advisory Board, Type Access, Repository for data, Impact factor."* (proposal, p. 3)
   This does not seem to make much sense in this stage of the project considering the shift in focus, see above 1.1 item 3, but we shall do it later on in the project.

2. Ad 1.2: **repositories**. The project plan was based on the idea of making an overview of journals (or their publishers) with their preferred repositories (see for example the *Journal of Open Archaeology Data* published by Ubiquity Press[4], which lists as repositories: ADS, DANS, Figshare, Open Context, tDAR, UCL Discovery). Since we do not gather details about the journals, the list of possible repositories becomes almost endless. The report has already listed the basic categories of repositories relevant to economic history (see section 4.1, p.

[1] L. Breure & M. Hoogerwerf, *Data Availability Policies: Ideal and Practice* (The Hague 2013).
[2] *International Review of Social History*: http://www.iisg.nl/irsh/
[3] *Tijdschrift voor Sociale en Economische Geschiedenis*: http://www.tseg.nl/index.php
[4] *JOAD*: http://openarchaeologydata.metajnl.com/repositories/

18-19), namely 'national', 'university' and CESSDA repositories. A list of recommended repositories will be provided.

3. Ad 1.3 and 2.1: data depositing practices and **impact of a DAP**. Nevertheless, we need a general model of journal workflow and data depositing practices in order to specify adequately the required data structure and functions of the demonstrator. Because we have chosen a simple DAP with a non-mandatory data review policy the overall impact will be low and we may rely on a *general* model of editorial workflow. The repositories themselves are only slightly affected by the DAP in that they have to notify the journal editors that a data set has been actually received.

1.3. About this document

This report contains the system analysis and design of the demonstrator, a broker application that acts as intermediary between authors, journals and the (preferred) repository, being DANS. The analysis of the information exchange between these three parties provides the basis for the second element of the report's title, the design. Designing software implies modeling and any model is a simplification of reality. At this stage the design is more conceptual, rather than technical. At first we want to reach agreement on the assumptions on which the model is based. Most failure of applications is due to too limited views and unilateral approaches.

> *Therefore, this report must not be considered as a guideline for developers only, but should be thoroughly discussed and approved by all stakeholders! The development of a correct and adequate application is a shared responsibility.*

The design of the demonstrator will comprise the following steps:

1. We shall start with a rough sketch of the collection of actors, objects and activities considered as relevant to the application. This functions as a demarcation of the "universe of discourse" , i.e. the world *as perceived and discussed in our project* (see chapter 2)
2. Next, we shall analyze the workflow of journals and repositories (see chapter 3).
3. Because the application will run at the repository side, we shall focus on the repository workflow and derive from this the application's functions (i.e. the functionality of the broker). A list of functions with their description is to be found in chapter 4.
4. Finally, we have to define on what data these functions will operate. This results into a ground plan of the database, formally defined as a so-called entity-relationship model (see chapter 5).
5. A very short summary and follow-up is to be found in chapter 6.

For the graphical representation of the design we shall mostly use UML diagrams. UML is the acronym of the Unified Modeling Language, which is since 2000 a standard in software design. It is a standardized, general-purpose modeling language in the field of software engineering, which includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies and offers a standard way to visualize a system's architectural blueprints. Although it is standard and widely used, not all UML diagrams are easy and intuitive to understand. Therefore, sidebars with some basic explanation have been added to make the diagram more legible.

The technical design will be part of the next work package, which will deliver the first implementation, and will be developed as by-product of the actual programming in an iterative way. Afterwards these technical models can be also used as system documentation.

1.4 Changes in version 1.2 of this document

The discussion of version 1.1. made the following updates necessary:

1. Section 1.2 **Implications** – *added:*
    - "…but we shall do it later on in the project."
    - "A list of recommended repositories will be provided."

2. Section 2.2 **Activity model** – *added:*
    - to diagram: notification at publication.
    - to text: textual summary of the diagram.

3. Section 3 **Workflows** – *added*:
    - diagram Ubiquity Press
    - more explanatory comments on the workflow diagram:
        - identifiers explained
        - 'Curate' also implies making data available to reviewers in case a data review is part of the article review itself.
        - 'Publishing' means making data available to everyone taking into account any restriction in force.

4. Section 4 **Application functions** – *added*:
    - creating special account for reviewers
    - some minor text corrections and clarifications.

5. Section 6 **Summary and follow-up** – *added*:
    - disclaimer

2. Overview

2.1 Actors, goals, responsibilities

The demonstrator under development concerns the interactions between author, journal and repository with regard to submitting an article for publication, together with the data on which it is based. The latter will be stored in a repository. It must take care of the responsibilities of all parties involved such as registration and authentication of the author(s), the quality of the data submitted (at least in a formal sense), the exchange of information between journal and repository etcetera. All entities involved, together with their dominant goals and responsibilities, are summarized in the mind map below (figure 1). This intuitive diagram is a representation of the universe of discourse in this project and, therefore, a guideline for what is included in in the design (for example, we disregard the role of funding agencies, the institutions where authors are employed, publishers' business models and comments through social media – to mention a few elements that are not modeled here, but in the real world may influence the actual process of publishing and data production).

We proceed from the assumption, that the publication is sent to the journal and the data are directly submitted to the repository, which requires some communication between both of them to effectuate the DAP. The journal has recommended the repository (in this case DANS) to the author and has to send a notification that a submission from author X for article Y is to be expected. The repository will invite the author to submit, for example by sending an email comprising a link that opens a submission screen. When the data have been successfully submitted, the repository has to inform the journal that the submission has been completed, which implies green light for publishing the article. Some journals may follow a different policy and may want to receive both, article and publication, but even in that case we assume that the author himself will deposit the data.

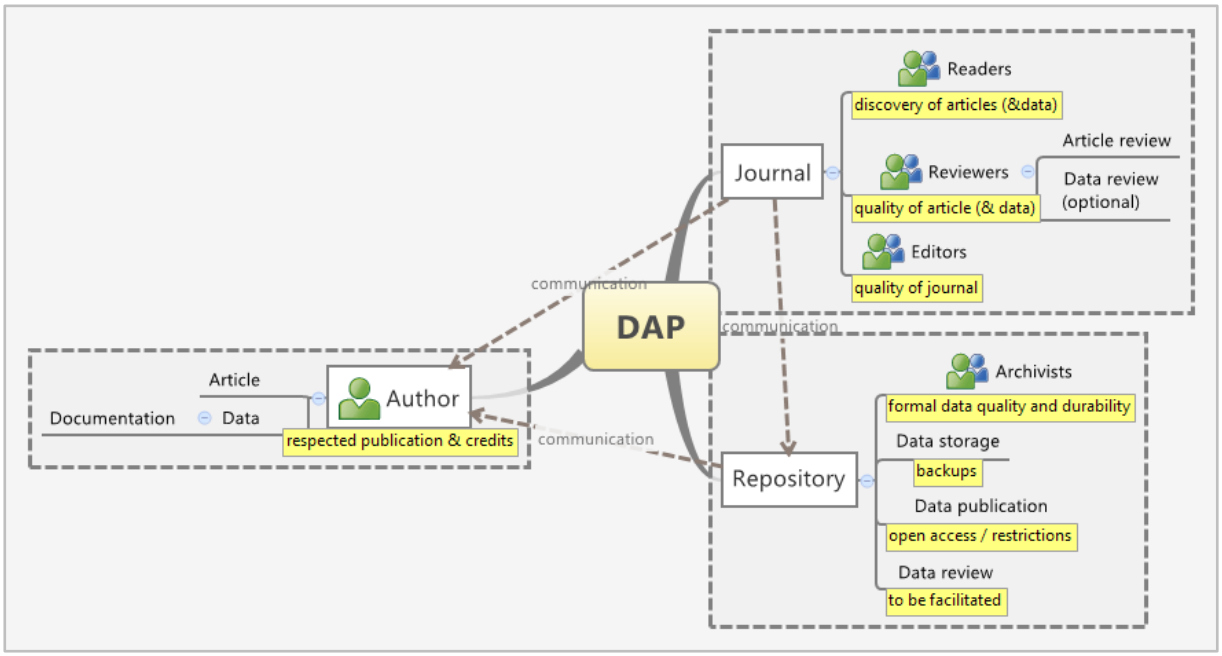


Figure 1: Mind map of concepts associated with a data availability policy (DAP).

2.2. Activity model

Goals and responsibilities will lead to activities and exchange of information between the parties involved. The system will provide a variety of broker services. Before diving into details it is helpful to make a *global* sketch of how the different interactions depend on each other, which can be visualized in the form of a so-called *use case diagram* (figure 2), which is the next step towards a more detailed design.

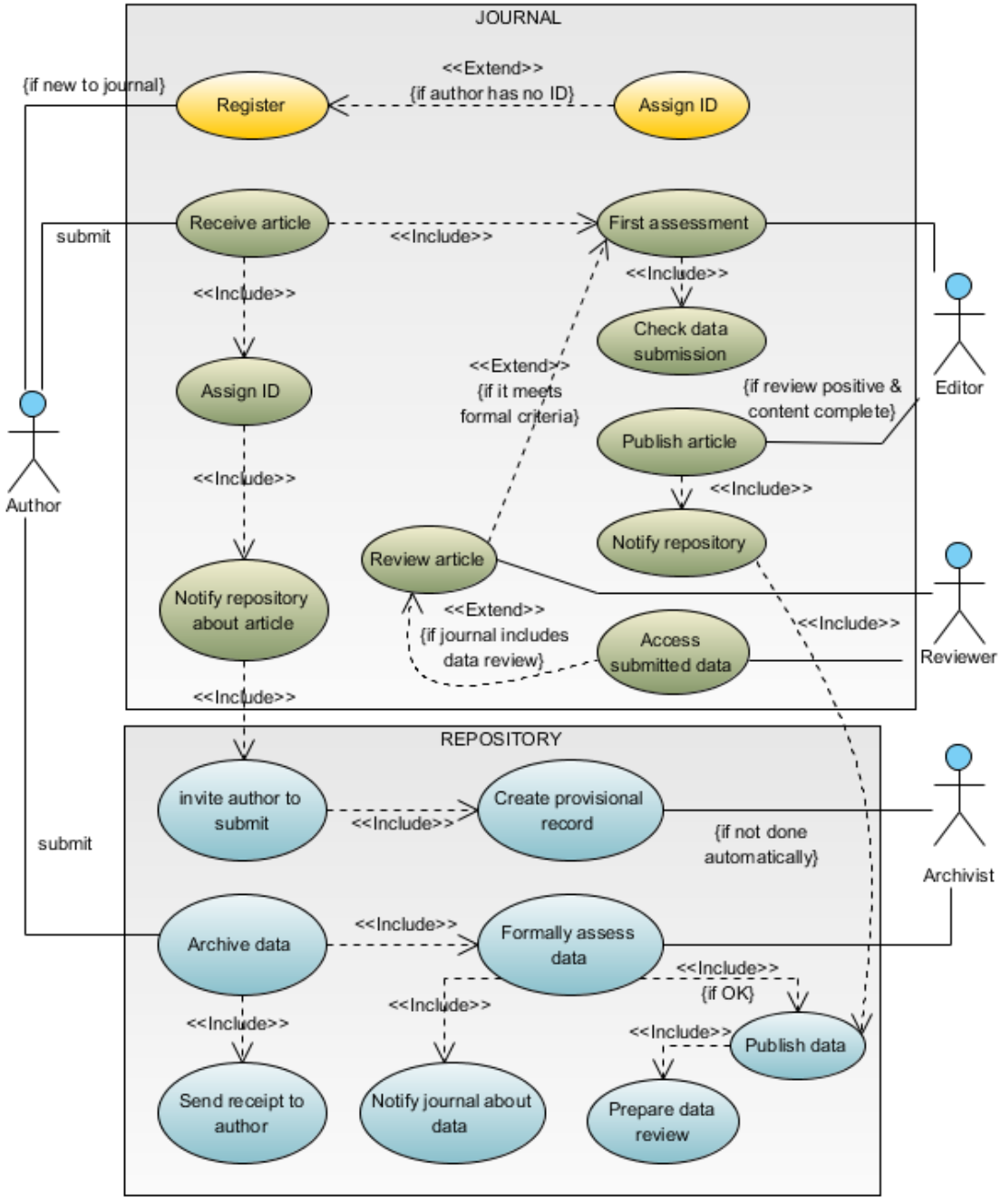


Figure 2: Overview of the CLIO-DAP broker activities in the form of a use case diagram.

A single *use case* represents an individual interaction between an external actor and the system. Or, in terms of gathering requirements: a use case is a contract of interaction between system and actor. An actor might be a person, a company or organization, a computer program, or a computer system

— hardware, software, or both. A use case should stimulate a discussion about what a system should do, mainly with people who are outside of the development team.

Use cases can be integrated into a *use case diagram*. While a use case description (omitted here) might drill into a lot of details about every possibility, a use case diagram can provide a higher-level view of the system.

A use case diagram provides a *simplified* and graphical representation of what the system must actually do at a high level of abstraction. See the sidebar 'How to read a use case diagram' for further explanation of symbols and interpretation.

A textual summary of the activities is:

1. The journal receives an article, does a formal check, and, if this is OK, asks reviewers to make an *article review*, which may include a *data review* as well, however, the latter is not mandatory in our plan.

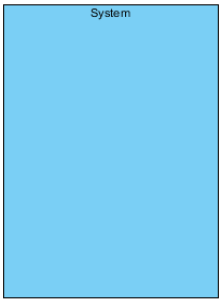
2. The journal notifies the repository that it may expect a data submission. The repository prepares a submission record, notifies the author that he may deposit data (typically through an email containing a link to the data record). The author will deposit his/her data.

3. If the data submission meets formal requirements, the data archive notifies the journal that data have been submitted. Data are now available only to potential reviewers asked by the journal and to the author.

4. When the journal has actually published the article, it notifies the repository, which makes the data now available to everyone and prepares a template for a data paper (to be completed by the author), which will be the basis of an optional data review by the scholarly community.

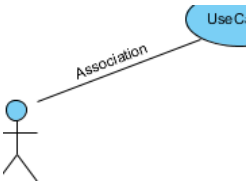
How to read a use case diagram

Please, note that use case diagrams are quite different from flow charts: they do not show a necessary sequence of actions, but show that 'if you do A you *must* also do B' (i.e. «Include») or 'if you do A you *may* do B' – under certain conditions (i.e. «Extend») without specifying when and in which order things are done.

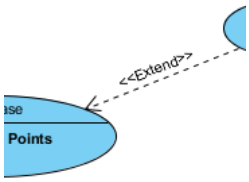


A system boundary indicates that the included use cases apply to that system, in this case the journal or the repository.

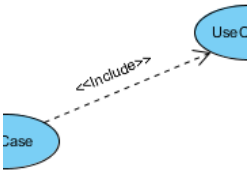
A use case is the specification of an action performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.



An association between an actor (stickman) and a use case (action) indicates who performs the action.



The extend relationship specifies that the behavior of a use case may be extended by the behavior of another (usually supplementary) use case.



The include relationship: the included use case is inserted into the behavior of the including use case. Note that the included use case is not optional, and is always required for the including use case to execute correctly.

3. Workflows

3.1. Introduction

In this section we shall describe the aforementioned actions in detail by analyzing the workflows journals and repositories as well as possible, but on the web not much is to be found about the internal activities of both. Therefore, we have to rely on common knowledge, exceptional insights in how work is organized and on our own knowledge of a data archive. The *Journal of Open Archaeology Data* (Ubiquity Press) has nicely summarized in the figure 3 below what happens when a data paper is submitted.

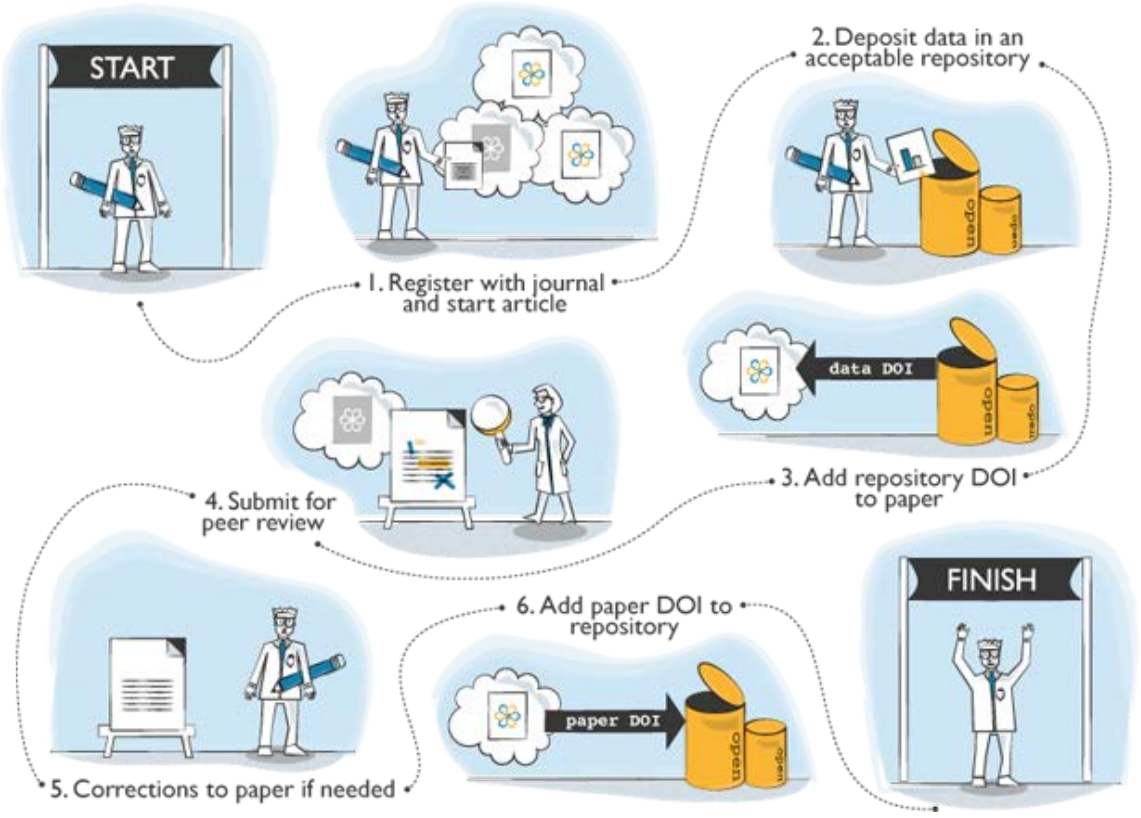


Figure 3: Summary of workflow with *JoAD*.

3.2. Editorial workflow

Most information about the workflow of journals is limited to guidelines for authors, which contain many details about fonts, margins, bibliographic references etcetera. One of the rare exceptions is the standard editorial workflow described for the *Lethbridge Journal Incubator*, which is an initiative of the University of Lethbridge Library and the University Of Lethbridge School Of Graduate Studies. The goal of this incubator is to address the issue of the sustainability of scholarly communication in an open access, digital age by aligning it with the educational and research missions of the university.

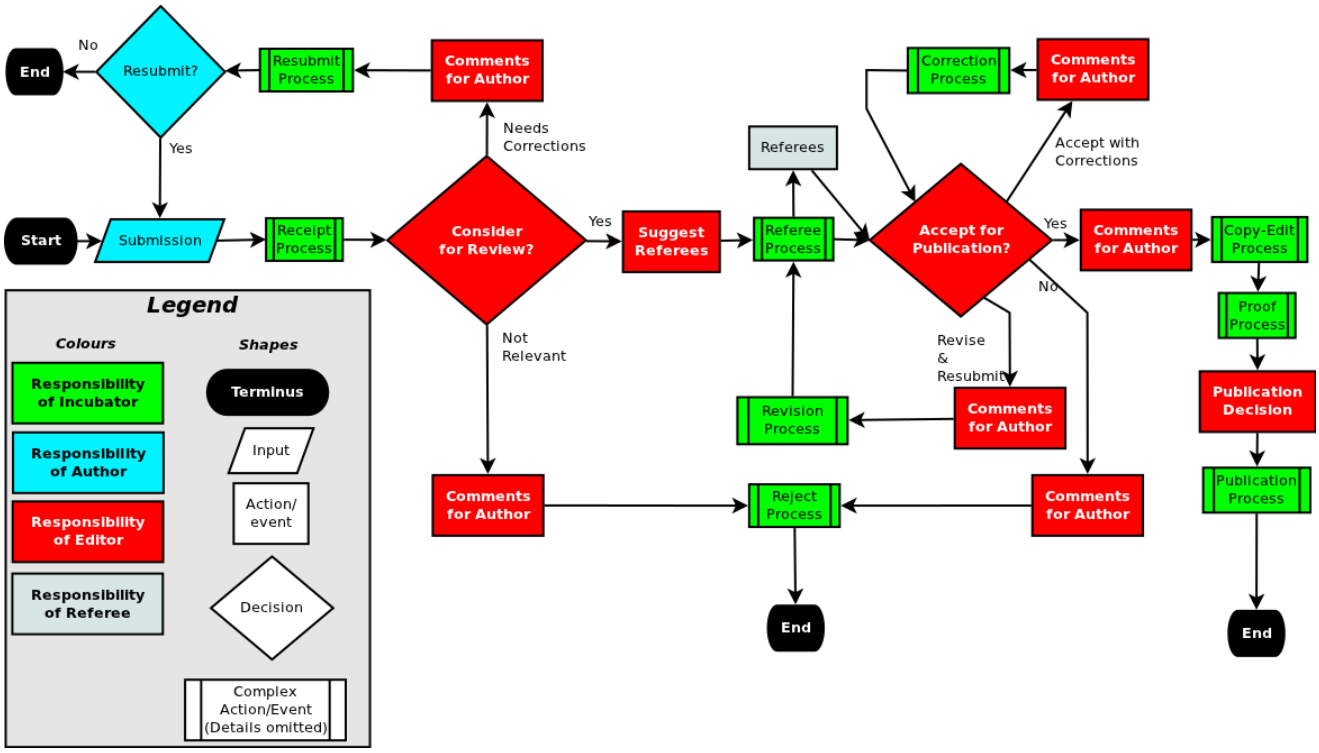


Figure 4: Standard editorial workflow in the *Lethbridge Journal Incubator*.

The basic premise of the incubator[5] is that the skills and experiences involved in contemporary scholarly journal production are both generalizable across disciplines and of significant educational value to graduate students. Through their work in the incubator, students will acquire training, managerial experience, and networking opportunities that are both of immediate use to them in their research domains and easily transferred to other aspects of their academic or professional careers. Each journal in the incubator adapts the workflow to match their specific needs and existing processes. Sections in red are the responsibility of the scientific editors; sections in green the responsibility of the incubator. The general nature of this design makes it useful for our purpose.

A similar, slightly more detailed flow chart is to be found with the *Intellect Journals*, an open access publishing house with journals on agriculture, biotechnology and medicine[6]:

[5] Lethbridge Journal Incubator: http://www.uleth.ca/lib/incubator/proposal.html, image at: http://www.journalincubator.org/the-incubator-workflow/

[6] Workflow Intellect Journals: http://www.intellectbooks.co.uk/MediaManager/File/peerreview.pdf

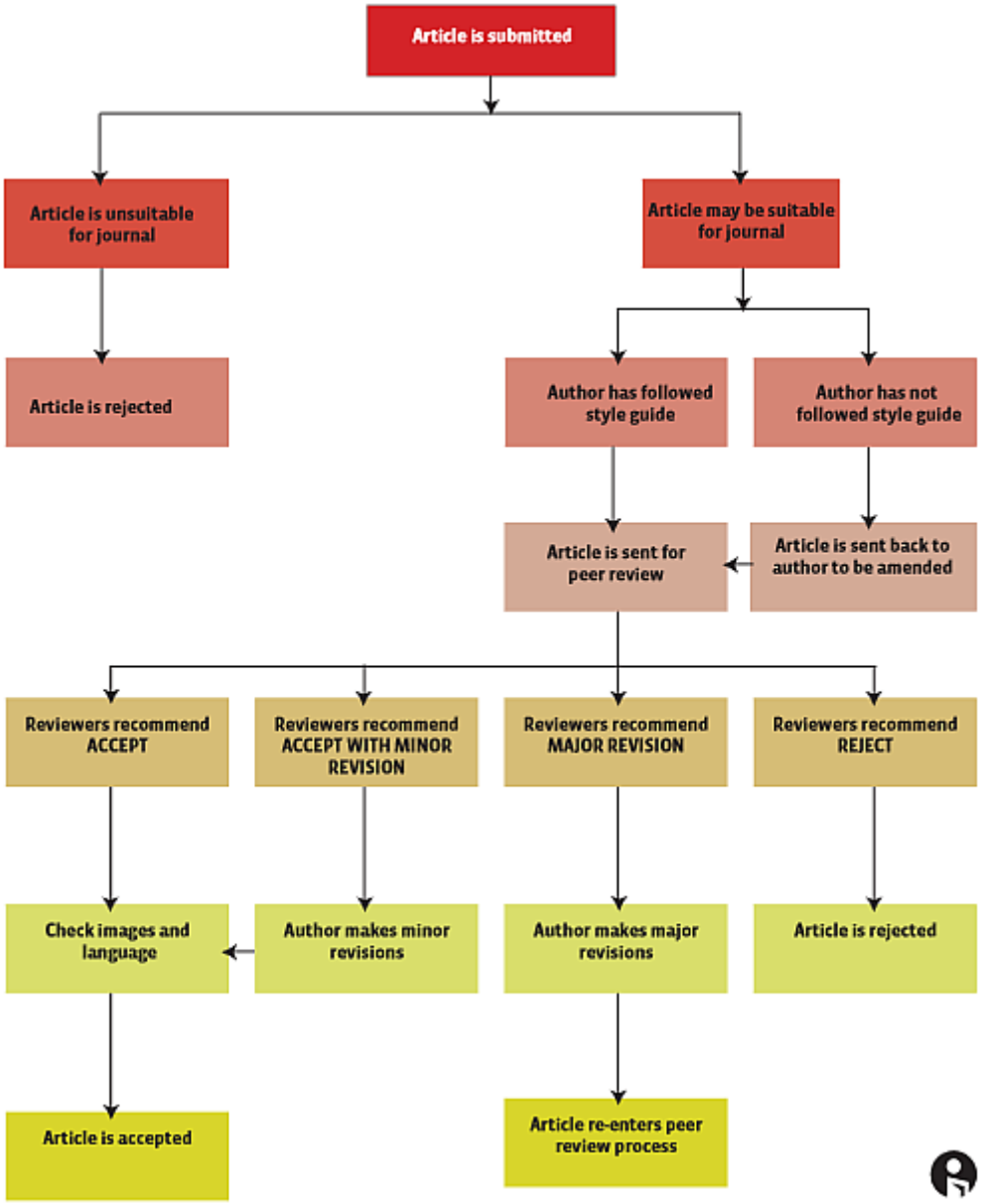


Figure 5: Editorial process with *Intellect Journals*.

In summary, editorial workflow as described in these two diagrams comprises the following essential steps, which can be easily extended with data submission, which we need to add in our case:

1. Submission of article (to the journal) and data set (to the repository) by the author. The system should send the author a receipt. This will be done automatically if a submission tracking system is used.
    - The repository must send a notification to the journal declaring that the data have been submitted and that they comply with the formal requirements of the data archive.

2. Preliminary evaluation of the article by the editor: does the article meet basic requirements?
    - The editor must also check that the notification from the data archive has actually arrived.
3. Review by referees.
    - Referees should be notified where the data have been deposited and be given access to them.
4. Decision on basis of the review: acceptance, revision (with iteration of the workflow) or rejection.
    - A special case occurs when the article is finally rejected. The repository should be notified of this fact and must have a rule about handling deposited data in such a case.
5. Final check before publishing.
6. Publication of the article with reference to the archived data set.

This workflow implies several different, successive states which are passed through by a publication. These 'submission states' are summarized in the state transition diagram below and would be tracked by a submission system.

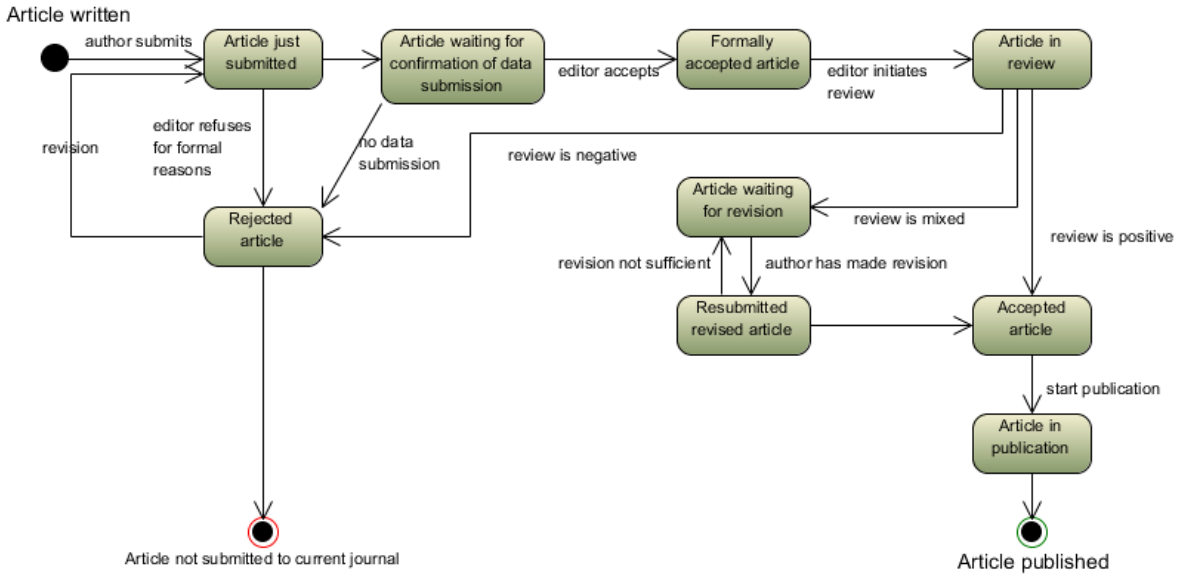


Figure 6: Different states of a scholarly article in the submission process.

3.3. Repository workflow

Most repositories publish guidelines for depositors, such as the fact sheet of DANS[7] below (figure 7), but don't mention their internal procedures. Behind the screens at DANS the following actions will take place, which we suppose to be similar to what is done elsewhere. After data have been deposited, an archivist processes the data set according to a standard protocol. This protocol aims at securing the long term discoverability, accessibility and understandability of the data without the intervention of the researcher himself. An important part of the protocol is the control on privacy sensitive information. This applies in particular to survey data and interviews.

[7] See: http://www.dans.knaw.nl/sites/default/files/file/factsheets/Factsheet%20Your%207%20steps%20to%20sustainable%20data%20DEF.pdf

On the basis of this protocol the following checks are performed:

1. for *completeness* of the data submission, which includes both, the deposited data files, the documentation files and the metadata (Dublin Core[8]) added by the depositor; an accompanying publication will get a Digital Object Identifier (DOI);
2. on the *readability* of the files; can all files be opened?
3. on the file *format* using a list of preferred file formats, whose durability will be guaranteed by the data archive;
4. on the *description* of the data set for completeness and accuracy;
5. on sensitive information in both, the data files and in the metadata; where applicable sensitive data are removed or data are anonymized; check on licenses for open or restricted usage;
6. on the clarity of the *directory* structure.

During this archiving process a persistent identifier is automatically generated and assigned to the data set, which allows others to refer to the submitted data without using fragile web addresses.

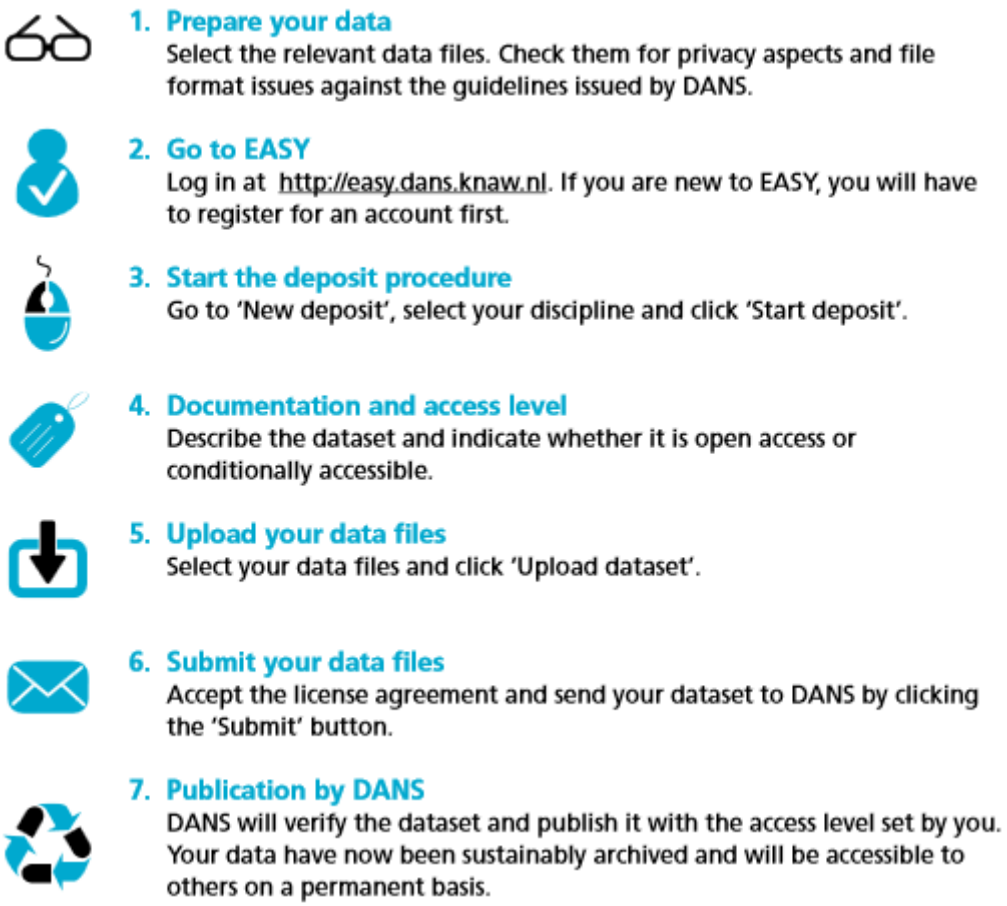


Figure 7: Data depositing steps with DANS.

[8] Dublin Core: http://dublincore.org/

Essential documentation files are:

- A description of the variables in the database(s) or spreadsheet(s) and an explanation of the codes used, in the form of a code book.
- A description of the (archival) sources, the selection procedure used, and a description of the way in which the archival sources have been used. The depositor should indicate whether the information has been standardized, and which standards or classification systems have been used, such as HISCO[9] for historical professional titles.
- All research and project reports relating to the research project and the dataset.
- All (academic) publications which originate from your research, and relate to the dataset.

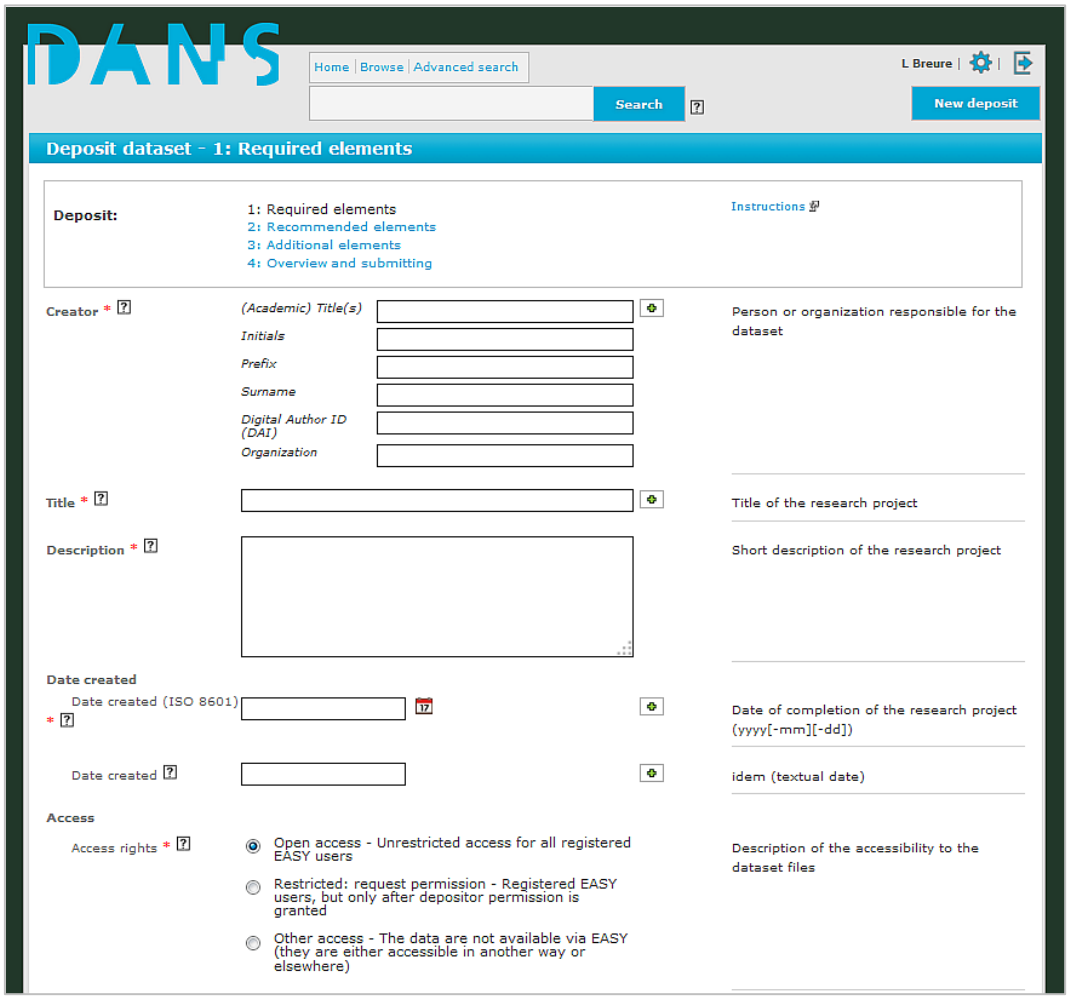


Figure 8: Depositing screen of DANS EASY.

[9] HISCO: http://historyofwork.iisg.nl/

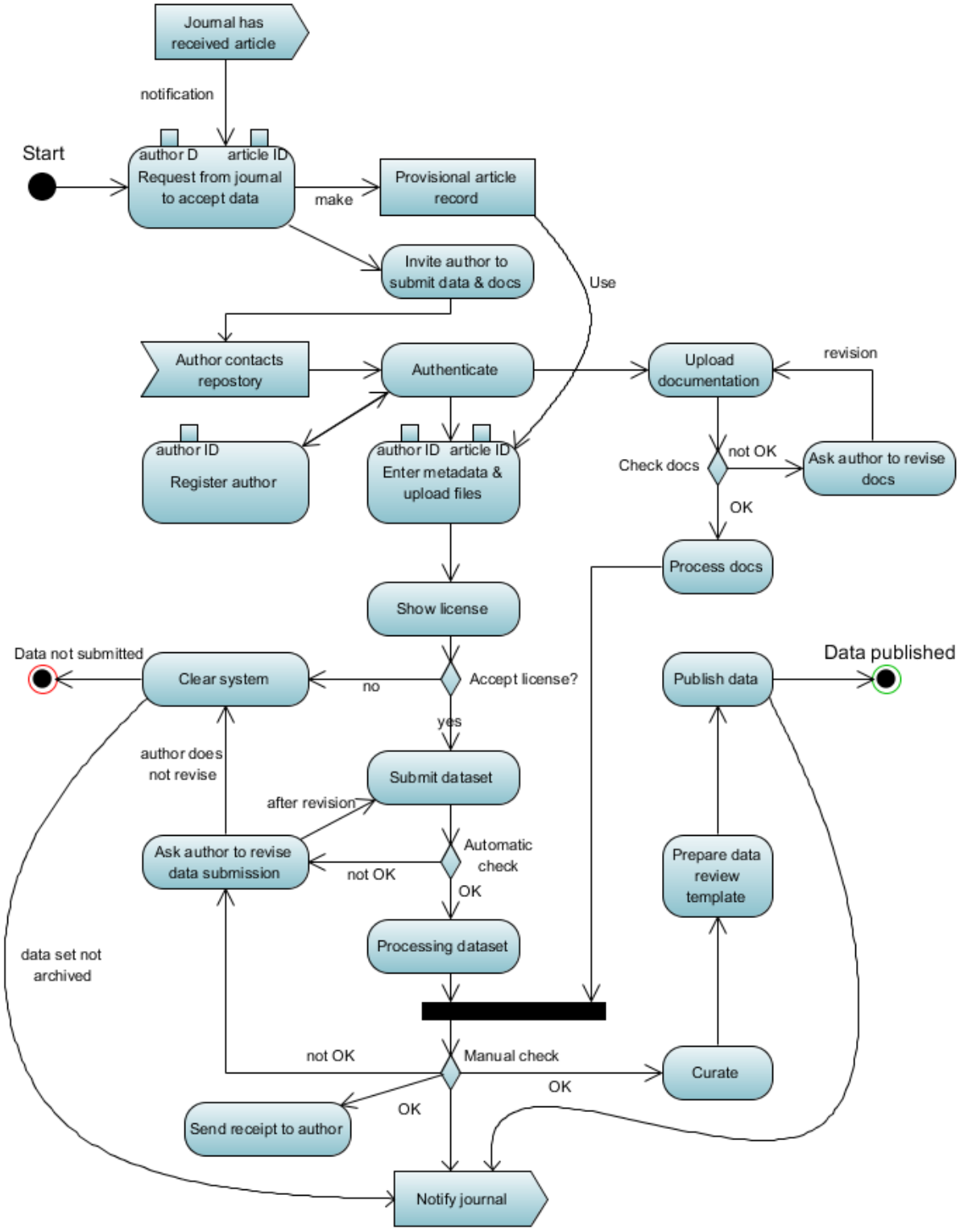


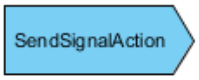
Figure 9: Repository workflow represented in an activity diagram.

In our case, this outline of more or less standard archiving activities has to be extended with the communication with the journal that has accepted DANS as preferred repository:

1. The journal must notify DANS that a publication has been submitted by author X. Note that the following *standard identifiers* can be used:

   a. for author: DAI, ORCID
   b. for article: DOI, handle
   c. for data set: DOI, URN, handle
2. DANS will invite author X by email (preferrably including a link to the prepared data record) to submit the data concerned.
3. If the data set has been submitted and formally approved, DANS will send a message to the journal, that author X has fulfilled all requirements.
4. DANS will make the data set available to potential reviewers asked by the journal (in case the journal includes data review as part of the article review) – here in the diagram part of 'Curate'.
5. DANS will prepare a template for a data paper to be completed by the author containing all administrative data as registered during archiving. The data paper is based on criteria that are later also to be used by data reviewers.
6. When the article is actually published, DANS will make the data available to all users (if there are no restrictions on openness). The journal must notify DANS about publication.

Taking this together leads to a detailed repository workflow as represented in the *activity diagram* in figure 9. We assume that the labeling of the actions is rather self-explanatory. The best way to describe the associated details is deriving the application's functions from this activity diagram. In the next section we shall go through the diagram and specify the function(s) corresponding with each activity and state the condition which must exist before a function can be used (*precondition*) and the result when a function is completed (*postcondition*).

How to read an activity diagram			
An activity diagram can be almost read as a flow chart and is, for that reason, a rather intuitive model. The most important difference is the use of symbols that match modern information technology.			
	An *initial node* is a control node at which the flow starts. A diagram may have more than one initial node.		*Final node*. An activity may have more than one final node. The first one reached stops all flows in the diagram.
	A rounded rectangle represents an *action*, i.e. an elementary behavior.		An *object node* indicates an instance of data, e.g. a data record.
	An *AcceptEventAction* is an action that waits for the occurrence of an event meeting a specified condition.		A *SendSignalAction* is an action that creates a signal and transmits it to the target object.
	A *decision node* represents a choice and has alternative outgoing flows.		A *join node* is a control node that synchronizes multiple flows. A join node has multiple incoming flows and one outgoing flow.

As a consequence of the workflow described above a data submission (in broad terms) has also different states, similar to the states of an article submitted to a journal. For the sake of completeness these states are summarized in the *state transition diagram* below.

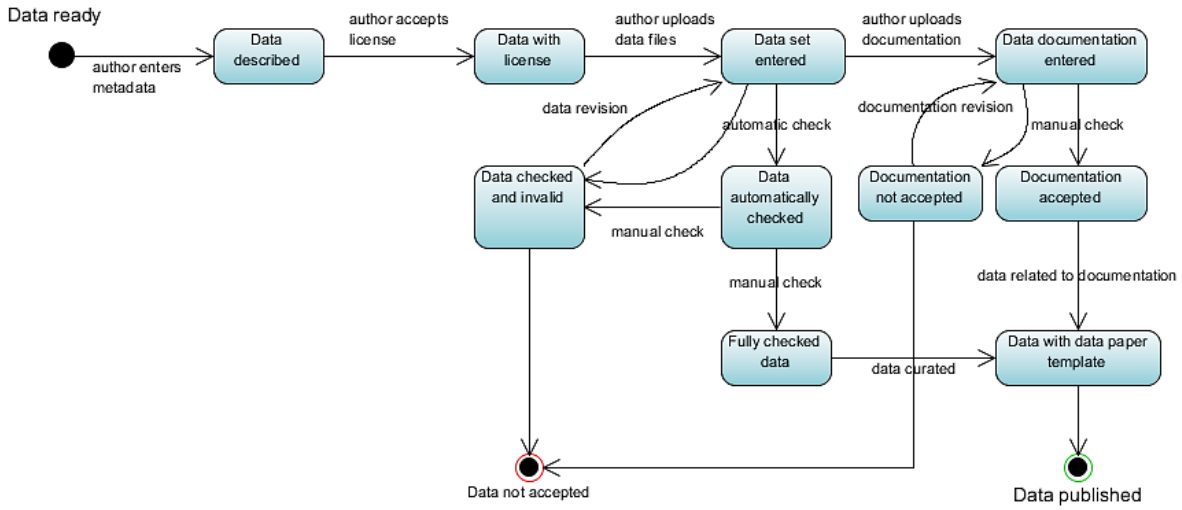


Figure 10: Different states of a data submission.

4. Application functions

Based on the activity diagram in figure 9 the following functions can be specified. Currently a few elements are still unsure or are to be negotiated with the journals involved (text red). Some functions are not explicitly shown as actions in the diagram, but logically implied and therefore listed below.

Functions must also take care of exceptional situations in the workflow. An *exception* is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. Exceptions should be adequately handled by the application.

Nr.	Function name	Description	Pre-condition	Post-condition
1.	Register journal	Assign identifier and collect and store contact information about journal	Journal has communicated to collaborate with DANS as a repository.	Journal's details are stored.
2.	Processing journal request	Receive and store notification. Prepare system for data ingest. Exception: journal does not pass identifiers – assign identifiers.	1. Notification from journal received (e.g. by e-mail). 2. Journal may or may not pass an authorID and articleID. [10]	1. Notification has been stored. 2. Archivist is informed. 3. Provisional article record has been created, including identifiers. 4. Author is invited to submit data using identifiers assigned.
3.	Authenticate depositor	Confirm the identity of the depositor, who should be (one of) the author(s) of the article and identify which article is concerned. Exception 1: depositor is not registered – let him register. Exception 2: a provisional article record is missing – ask depositor to submit article first.	1. A registered depositor contacts broker application through online interface. 2. A provisional article record is available.	1. Depositor's personal data are formally checked and stored. 2. Depositor is authorized to deposit, update and delete data.
4.	Register depositor	Standard user registration EASY, may include updating of personal information.	Depositor is unregistered.	Depositor is registered.
5.	Enter article's metadata	1. Entering all required metadata concerning the article, perform formal checks (format, mandatory). 2. Depositor accepts license. 'Entering' may imply 'updating'. Exception: depositor has not entered all required metadata and/or has not accepted license, so	Depositor is registered and authorized.	Metadata of an article are stored.

[10] See also p. 16: standard identifiers.

		submission failed – clear system.		
6.	Upload data files (submit dataset)	Standard data upload procedure of EASY. Upload may include update. Exception: data do not pass first formal check – send message to depositor to revise data.	Metadata have been entered.	1. Data have been uploaded and stored. 2. Data have been checked automatically. 3. Notify archivist that data set can be further processed.
7.	Upload documentation	Documentation about the data set is uploaded and formally checked. Upload may include update. Exception: documentation does not pass first formal check – send message to depositor to revise documentation.	Depositor is authenticated.	1. Documentation has been uploaded and stored. 2. Documentation has been checked automatically.
8.	Delete data and documentation [11]	User withdraws data set plus documentation or archivist has a reason to delete.	Data and/or documentation have been deposited.	1. Data and documentation are fully removed from the repository. 2. Journal is notified about deletion.
9.	Manual check [12]	Archivist inspects data and documentation and gives final approval to submission.	Data and documentation have been uploaded and checked automatically.	1. Submission is flagged as complete and approved. 2. Receipt of data submission is sent to depositor. 3. Journal is notified about successful submission.
10.	Create account for reviewers	If a journal includes data review as part of the peer review of an article, reviewers must have access to deposited data.	Submitted data have been approved.	Reviewers can download the submitted dataset.
11.	Publish data	Data set is made available online taking into account possible restrictions on openness.	Submission is flagged as complete and approved.	1. Data set and documentation are available online. 2. Journal is notified about data publication (journal may inform reviewers).
12.	Prepare data paper template	A standard data paper template is filled with required metadata and published.	Data submission is complete and approved.	A customized data review template is online available.
13.	Monitor workflow	Producing a report showing the progress of the submission, including exceptions which have	A submission has started.	Display of a log file of activities.

[11] This may be an manual action by the administrator.

[12] This may be implemented as part of 'Monitor workflow'.

		occurred.		
14.	Browse notifications [13]	Internal administrative function: search, retrieve and create a report about all notifications regarding a submission.	Archivist or administrator retrieves all or a specific category of notifications regarding an author or article.	Requested notifications are displayed in a required format.

[13] Typically registration of e-mails between all parties concerned.

5. Data model

5.1. Data model versus activity models

So far our focus has been on the *activities* in our universe of discourse, which was roughly defined in the mind map in figure 1. An important counterpart is the *data* we are going to record during these activities. These data do not only comprise the research data submitted by the depositor but all other administrative data as well. We have to indicate what storage is needed for all this information, in other words, we have to draw a 'map' of the database we want to have. Data modeling defines not just data elements, but also their structures and the relationships between them, which are essential for correct functioning of the application. As with workflow analysis we also make here important assumptions about how our 'world' looks like. These assumptions may be correct or not, and therefore, are worth to be explicitly stated and checked by all stakeholders. Like all models, a data model is also a simplification: we show only what we consider as relevant, and leave out all things that may cause superfluous complexity.

5.2. Different levels of modeling

A data model usually consists on different levels of abstraction:

1. A conceptual schema or *conceptual data model* is a map of concepts and their relationships used for databases. This describes the *semantics* of an organization and represents a series of assertions about its nature. Specifically, it describes the things of significance to an organization (entity classes), about which it is inclined to collect information: characteristics (attributes) and associations between pairs of those things of significance (relationships). A conceptual data model is essentially a set of technology independent specifications about the data and is used to discuss initial requirements with the business stakeholders.
2. The next step in data base modeling is translation of the conceptual model into a *logical* data model, which documents structures of the data that can be implemented in databases. Implementation requires that specific characteristics of the storage system (e.g. a relational database) are taken into account. A conceptual model may show a configuration that cannot be implemented.

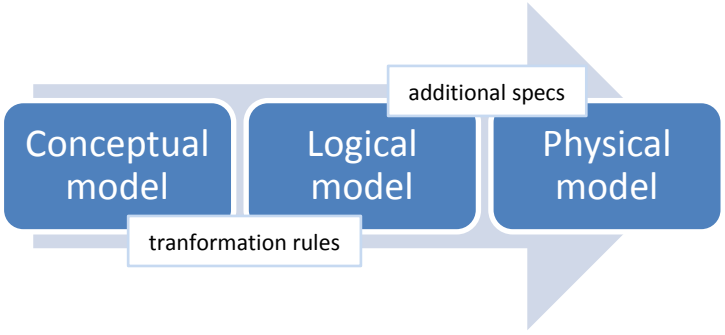


Figure 11: Sequence of database models.

3. The last step in data modeling is transforming the logical data model into a *physical* data model that organizes the data into tables, and accounts for access, performance and storage details.

In this report we limit ourselves to the conceptual model. The logical model is not worth to be discussed in a broader group. The physical model can be most efficiently created as part of the programming stage.

5.3. Entity-relationship diagram (ERD)

5.3.1. ERD: Entities and relationships

Different types of diagrams are used to represent data models. So far we have used UML diagrams where available. The UML class diagram is also frequently used as conceptual data model; however, we prefer the older entity-relationship diagram (ERD), which is clear and even more expressive. For the sake of a broad discussion we shall keep the model as simple as possible. Once we have agreed upon the underlying assumptions, technical details can be added.

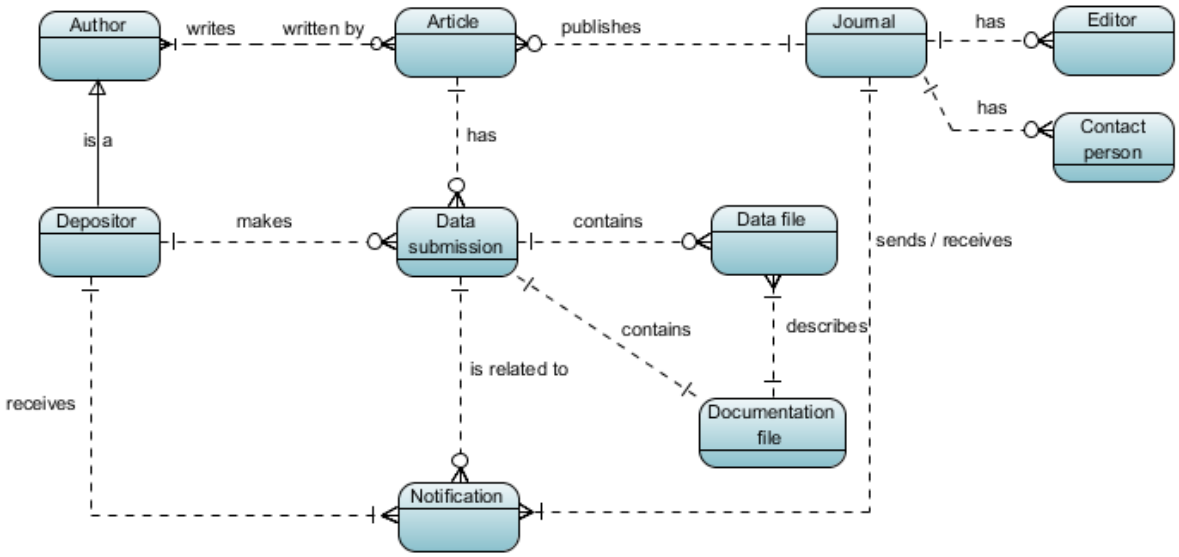


Figure 12: Basic entity-relationship diagram representing the data structure of the broker.

A textual description of the data model in figure 12 is the following:

1. We register the details about a journal, such as title, names of editors and data about one or more contact persons.
2. The depositor (being one of the authors) supply more details about the publication and its authors (we know already which journal is concerned).
3. A depositor can make one or more data submissions for the same article. More than one submission may be required because the supplied data and / or documentation may fail to meet our requirements.
4. A data submission[14] may contain one or more data files, which are described in a single documentation file. It is a matter of discussion whether we keep the subsequent submissions in the form of versions; probably we don't.
5. A submission is associated with many notifications, starting with the message from the journal that an article has been submitted, followed by the invitation to the author to submit data, and other messages as a result of checks (see activity diagram). It seems wise to keep a record of this information exchange; therefore, notifications are also stored in the database.

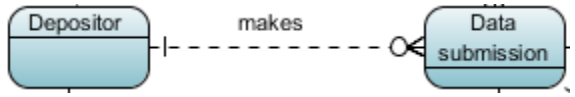
[14] A data submission is roughly equivalent to a dataset in EASY.

How to read an ERD?

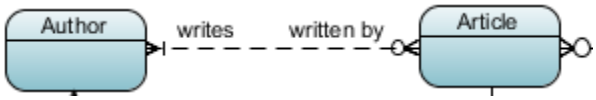
An entity relationship diagram (ERD) is a representation of data within a domain. It consists of *entities* as well as *relationships* between entities.

An entity can be a tangible, physical object such as a school or student, or a concept such as a reply or a transaction. In entity relationship modeling, the term entity has synonyms: "table", "database table", "entity-type". Yet, entity is the most commonly used term. Each entity brings along a set of columns, which are the properties of the entity the attributes belong to. For instance, entity Student has name, address and grade as columns (synonyms: attributes, properties, fields). Every entity must have at least one attribute that can be used to uniquely identify the entity, which is known as the entity's primary key.

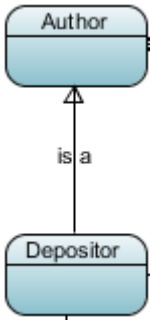
Relationships are capable in linking up entities. Typical examples: one-to-one (e.g. the data submission is supposed to contain one documentation file), one-to-many, or many-to-many. The proper use of relationship is important in showing HOW entities are related and specify what is valid or not. For instance, one-to-many relationship must be used for modeling the fact that 'one school has many students'.



One-to-many relationship: a depositor may make zero, one or many data submissions (a state of zero submissions is assumed to be valid in the system, although it may not occur often). A certain data submission is always made by one depositor. Notice, the symbols (bar, circle and crow foot: 1, zero, many).



Many-to-many relationship: an author (as far as the system knows him) may have written zero, one or many articles. An article must be written by at least one author, but many authors are also allowed.



Subclass: a depositor must be one of the authors of an article. A subclass indicates something special, mostly the recording of extra attributes than we do with the superclass.

5.3.1. ERD: Attributes (fields)

Entities are further described by their attributes (sometimes called 'data element', or 'fields' in terms of a physical database). These are the smallest units of data that can be described in a meaningful manner. Collectively, attributes define an entity. An attribute is meaningless by itself. For example, date of birth comes from the context of the entity to which it is assigned, for example, date of birth of an employee.

Attributes have a looser description than the data elements in a physical data model. For instance, whereas an attribute may have only a descriptive name, a data element needs:

- a size and range,
- a format and length,
- an accurate and detailed description,
- valid values

- defined edit rules.

All these details are postponed to the physical model.

A few attributes fullfil a special role of identifying entities and relating one entity to another: the *primary key* and *foreign key*.

- The **primary key** of an entity unambiguously distinguishes between occurrences of the entity. It is a unique identifier comprised of one or more attributes of the entity. Each occurrence of an entity has one value for its primary key. In the table below primary keys are underlined.
- **Foreign keys** (also known as referential attributes) are attributes that define relationships between entities. A foreign key in one entity corresponds with a primary key in another entity. For example, JournalID is the primary key of the entity JOURNAL and *JournalID* appears as foreign key in the entity ARTICLE, in this way associating an article to a journal.
- Attributes of which the name if followed by a * may occur more than once (a so-called repeating group).

Attributes in this data model

Author
- AuthorID
- TermOfAddress
- FirstName
- MiddleName
- LastName
- Institute
- Country
- EmailAddress

Depositor
(all Author attributes)
PhoneNumber

Journal
- JournalID
- JournalTitle
- Discipline *
- Publisher
- TypeOfDAP
  - mandatory
  - optional
  - undecided
- PreferredRepository
- AlternativeRepository *

Editor
- EditorID
- *JournalID*
- TermOfAddress
- FirstName
- MiddleName
- LastName
- Institute
- Country
- EmailAddress

ContacPerson
- CPersonID
- *JournalID*
- TermOfAddress
- FirstName
- MiddleName
- LastName
- Institute
- Country
- EmailAddress
- PhoneNumber
- SkypeAddress

Article
- ArticleID
- ArticleTitle
- ArticleSubtitle
- *AuthorID* *
- *JournalID*
- Volume
- Issue
- Pages
- Tag *

DataSubmission
- SubmissionID
- *ArticleID*
- *DepositorID* (= an AuthorID)
- Creator
- Contributor
- TypeOf Resource
- Genre
- License
- DateCreated
- DateIssued
- DateSubmitted (incl. time)
- Description
- Topic * (field, discipline)
- Temporal (about which periode)
- Relation (to other data and publications)
- Location (about which place)
- AccessCondition (open / restricted access)

Datafile
- FileID
- FileName
- FileType
- FileSize
- FileDate
- *SubmissionID*

Note: 'Datafile' represents the administration of the actual research data files, in other words, it contains metada.

Documentation
- DocumentationID
- DocFileName
- DocFileType
- DocFileSize
- DocFileDate
- *SubmissionID*

Note: 'Documentation' represents the administration of the actual documentation, in other words, it contains metada.

Notification
- NotificationID
- NotificationDate
- NotificationTime
- NotificationType
- Sender
- Receiver
- Content
- *DepositorID*
- *JournalID*
- *SubmissionID*

As mentioned above, this is a conceptual representation of data structure that the broker application requires and which is to be used to test and to check the goals and assumptions in this design. For practical reasons, for example because of existing repository software and infrastructure, the actual technical implementation may be different in many respects.

6. Summary and follow-up

In this report we have developed a conceptual design for a web application that supports data submission and the related information exchange between authors, journals and DANS (in the role of preferred repository) as part of the implementation of a DAP system. Starting from a rough sketch of the universe of discourse we have defined the workflows for both journals and repositories and defined the functionality of the application in more detail. We have concluded with a conceptual data model, comprising all entities of which we want to record data. The data themselves have been preliminary defined as attributes of these entities, postponing definitions of data type, field length and other technical details to the next stage of programming. Most importantly we should pay ample attention the underlying assumptions, which are the basis of the perspective from which we look at the application under development, and be aware of the possible limitations of our own views.

When this report has been fully discussed with all stakeholders, implementation will start. In small iterations we shall build the demonstrator and test it with data, first coming from DANS and, hopefully in short time, also from the journals that want to participate in a pilot.

During implementation more detailed technical documentation will be developed. This will include at least:

- A *physical database design* in the form of a detailed ERD, with description of fields insofar their meaning is not obvious and/or they are not fully documented elsewhere.
- A *documentation of the program* code by means of Javadoc or a similar tool. Javadoc is a documentation generator from Oracle Corporation for generating documentation in HTML format from Java source code. The HTML format is used to add the convenience of being able to hyperlink related documents together.
- Technical documentation about *installing and running* the application.

Disclaimer

We must emphasize that the result of this project is an application intended to demonstrate a proof of concept for a limited time and a limited number of journals and publications, not yet a fully operational service offered by DANS. Maintenance will be limited to bug fixing and does not include extension of functionality. At the end of the exploration period DANS will evaluate the demonstrator with all parties involved and decide jointly how the service can be continued on a more permanent basis.