ARTICLE

Stateful Library Analysis and Migration System (SLAM)

An ETL System for Performing Digital Library Migrations

Adrian-Tudor Pănescu, Teodora-Elena Grosu, and Vasile Manta

ABSTRACT

Interoperability between research management systems, especially digital libraries or repositories, has been a central theme in the community for the past years, with the discussion focused on means of enriching, linking, and disseminating outputs. This paper considers a frequently overlooked aspect, namely the migration of records across systems, by introducing the Stateful Library Analysis and Migration system (SLAM) and presenting practical experiences with migrating records from DSpace and Digital Commons repositories to Figshare.

INTRODUCTION

Bibliographic record repositories are a central part of the research venture, playing a key role in both the dissemination and preservation of outcomes such as journal articles, conference papers, theses and dissertations, monographs, and, more recently, datasets. As the ecosystem of which these are a part of has evolved at a sustained pace in the last decade, repositories also had to adapt while ensuring uninterrupted service to the research community. Nevertheless, a number of developments, both at the local, repository level and at a more general, global scale, have created the necessity of considering the complete replacement of certain systems with new repository solutions which are better suited for their stakeholders' requirements. The following are a few such developments:

- The need to consolidate both technological solutions and operational teams, in order to reduce running costs and provide a unified experience for end users, the research personnel.[1]
- Various policies require researchers to provide not only traditional outputs, such as journal articles or conference papers, but also the datasets and other materials backing up scientific claims. For repositories, this means both adapting to larger amounts of stored data as well as ensuring that the metadata dissemination and preservation mechanisms are suited for the new output types (e.g., while full-text search is a common feature of literature repositories, it cannot be easily applied to numeric datasets).[2]
- Apart from extending the set of stored outputs, policies have also created new requirements for existing record types. For example, the Research Excellence Framework (REF) in the UK mandates monitoring open access (OA) publishing of research articles; thus, institutional repositories are no longer only a facilitator of *green* open access (*self-archiving* of records) but also a means of monitoring compliance.[3] This requires the implementation of new logic in existing repositories, which can frequently be difficult, especially when faced with legacy repository code bases or insufficient technological resources.

Adrian-Tudor Pănescu (tudor@figshare.com) is Software Engineer, Figshare. **Teodora-Elena Grosu** (teodora@figshare.com) is Software Engineer, Figshare. **Vasile Manta** (vmanta@tuiasi.ro) is Professor, Faculty of Automatic Control and Computer Engineering, Gheorghe Asachi Technical University of Iași, Romania.

https://doi.org/10.6017/ital.v40i4.12035

- Commercial, contractual, or leadership changes can also create the need to replace repository systems, due to uncertainty (see the acquisition of bepress by Elsevier) or preference for certain platforms.[4]

While these developments can generate the requirement to switch repositories in a very short span of time, such a venture needs to be properly planned and executed in order to ensure, on the one hand, that no records are lost or corrupted and, on the other hand, that minimal or no downtime is caused. Ideally, migrations would also be an opportunity to curate and enrich the existing corpus by consolidating and correcting bibliographic records.

Between 2018 and 2019 the research team has performed six digital library migrations from various source repository solutions (DSpace, Digital Commons, custom in-house built systems) to the Figshare software as a service (SaaS) repository platform. For this purpose, SLAM, an extract, transform, load (ETL) system, was developed and successfully employed in order to migrate over 80,000 records. This article describes the rationale behind SLAM, its design and implementation, and the practical experiences with employing it for repository migrations. A number of future enhancements and open problems are also discussed.

MOTIVATION AND BACKGROUND OF SLAM

In early 2018 Figshare started considering the suitability of its repository platform for storing content which is usually specific to *institutional* repositories (journal articles, theses, monographs), along with non-traditional research outputs (datasets or scientific software).[5] While feature-wise this was validated by its hosted preprint servers, a new challenge was posed, as stakeholders choosing to use Figshare as an institutional repository also had to transfer all content from their existing systems.[6]

Thus, in the first half of 2018, a first migration was performed, transferring records from a bepress Digital Commons (DC) repository (https://www.bepress.com/products/digital-commons/) to Figshare (https://figshare.com). From a technical point of view, a Python (https://www.python.org/) script was developed for this migration; this script parsed a comma-separated values (CSV) report produced by DC which contained all metadata and links to the record files.[7] Using this information, records were created on the Figshare repository using its application programming interface (API) (https://api.figshare.com). While this migration succeeded, the naive technical solution presented a number of issues:

- Difficulties with the metadata crosswalk: While a crosswalk was initially set up, mostly based on the definition of the fields in the source and target repositories' metadata schema, issues were discovered while migrating the records, mainly generated by inconsistencies in the values of the fields across the corpus. These issues were fixed on a case-by-case basis, in order to ensure a lossless migration, but it would have been preferable to surface them in the early phases, in order to have the migration script mitigate the issues in the final run.
- Running the migration procedure multiple times: The migration script followed mostly an all or nothing approach, which, at each run, fully migrated all records between repositories. This is undesirable, as there was a need to run the script only for those records that failed to migrate (due, for example, to metadata crosswalk issues). After the full migration was completed, there was also a need to apply only some minor corrections to records, without following the full procedure. This was not possible, since the script would recreate all records to migrate from scratch on the target repository, as it did not have any memory of

previous runs. This issue was also amplified by the fact that in the source repository records did not have any type of persistent identifier attached. Thus, additional scripts, which only performed the corrections, had to be developed.
- Ability to run the migration procedure with minimal supervision: Like most migrations, this instance considered a large number of records (over 10,000) and, ideally, the process would run with minimal supervision operators. While the script partially accomplished this, the need for better fault-tolerance and enhanced logging was identified.

Given the lessons learned from the initial attempt and the requirement that five additional migrations were to be completed between October 2018 and December 2019, a more robust alternative to the naive migration script was required. This alternative had to adhere to three design principles:

1. Reusability: The system should be usable for multiple migrations without extensive additions or modifications. Thus, it should be able to adapt to the workflows of multiple repositories, metadata schemas, and other concerns specific to each migration.
2. Statefulness: In software engineering, programs can either discard knowledge of past transactions or preserve it, allowing previous results and operations to be revisited. Migration systems benefit from a stateful architecture, as the system should be able to perform the same migration multiple times, without creating duplicate records on the target repository, while allowing for incremental record improvements with each run. Apart from allowing for corrections to be applied post-migration, this would also support the prototyping phase (where multiple test migrations are performed in order to validate the metadata crosswalks), that no information is lost, and other general workflow aspects.
3. Fault tolerance: The system should implement fault tolerance mechanisms at all levels, allowing it to run migrations of large corpora with minimal supervision and, at the same time, implement sufficient logging and exception handling to allow operators to identify and correct potential issues.

Several repository migrations are represented in the literature. In Van Tuyl et al., the authors describe the process of moving from a DSpace (https://duraspace.org/dspace) to a Samvera (https://samvera.org) system, while in the study from Do Van Chau records were migrated from a solution developed in house to DSpace.[8] Both instances offer valuable insight into the challenges posed by digital library migrations, especially at the level of bibliographic metadata; on the other hand, both works are focused mostly on a specific use-case and do not propose general technical solutions for other migrations. It is interesting to note that the migration presented by Van Tuyl et al. required two and a half years of work, while SLAM was employed to carry five migrations in 14 months.

The Bridge2Hyku toolkit (https://bridge2hyku.github.io/toolkit) is a collection of tools, including a module for the Hyku repository solution (https://hyku.samvera.org), aimed at facilitating the import of records into digital libraries based on this software. Similar to SLAM, it includes an analysis component, useful for surfacing and correcting potential metadata issues during the migration. SLAM provides two major improvements over this solution, namely it defines a generic architecture that can be used for migrating records between any two repositories, while also defining a procedural migration workflow to create a robust, fault-tolerant, and extensible solution.

Pygrametl (http://chrthomsen.github.io/pygrametl/) and petl (https://github.com/petl-developers/petl) are two open-source frameworks which allow the defining of ETL workflows; similar to SLAM, the processing steps are defined using Python functions. These projects are targeted towards tabular and numeric data, making them unsuitable for the transfer of files and metadata across bibliographic repositories.

Singer (https://www.singer.io/) is an ETL framework similar in design to SLAM, which allows the composing of various data sources (or *taps*) and targets, in order to move data between them. The two downsides of this implementation are that it is focused on processing data specified in the JavaScript Object Notation (JSON) format, which is not always available for bibliographic metadata, and that it does not facilitate extending the pipeline with, for example, the analysis facilities targeted by SLAM.

Hevo Data (https://hevodata.com/), Pentaho Kettle (https://github.com/pentaho/pentaho-kettle) and Talend Open Studio (https://www.talend.com/products/talend-open-studio/) are ETL frameworks which employ graphical interfaces to allow users to define the processing workflows. While such functionality was not initially identified as a requirement for our planned migration projects, during testing it became obvious that providing such an interface could bring value by having repository administrators be more involved in defining and validating the processing applied to bibliographic records, as the administrators possess the most knowledge of the organisation of the repositories. A downside of the three solutions is that their usage requires commercial agreements, which did not line up with the business requirements of the considered migrations.

In their work, Tešendić and Boberić Krstićev use the Pentaho suite in order to implement the ETL component of a Business Intelligence (BI) solution for reporting on bibliographic records.[9] While the structure of the ETL processing is different—the authors being mostly interested only on certain aspects of the metadata—this work provides insights into the types of analysis that could be performed while migrating records.

SLAM'S DESIGN AND IMPLEMENTATION

Following the design principles previously mentioned, SLAM's architecture was devised as presented in figure 1; as for most ETL systems, the easiest way of understanding its operation is by examining the data flow.

The migration workflow proceeds by extracting all the required information from the source repository. This could be achieved in multiple ways, such as harvesting through an OAI-PMH (https://www.openarchives.org/pmh/) endpoint or other types of API, using the bulk export functionality implemented by most repository systems, or even by crawling the HTML markup describing records, similar to what search engines do in order to discover web pages. Once this mechanism has been established, practical experience proves that it is beneficial to move this raw data *closer* to the destination repository (to a *staging* area as depicted in figure 1). While this transfer might prove cumbersome, especially for large corpora, it is required only once. Moreover, having the data close to the destination repository allows faster prototyping and testing of the migration procedure, as network latency and throughput are improved, while also ensuring that the source repository's functioning is not affected in any manner.

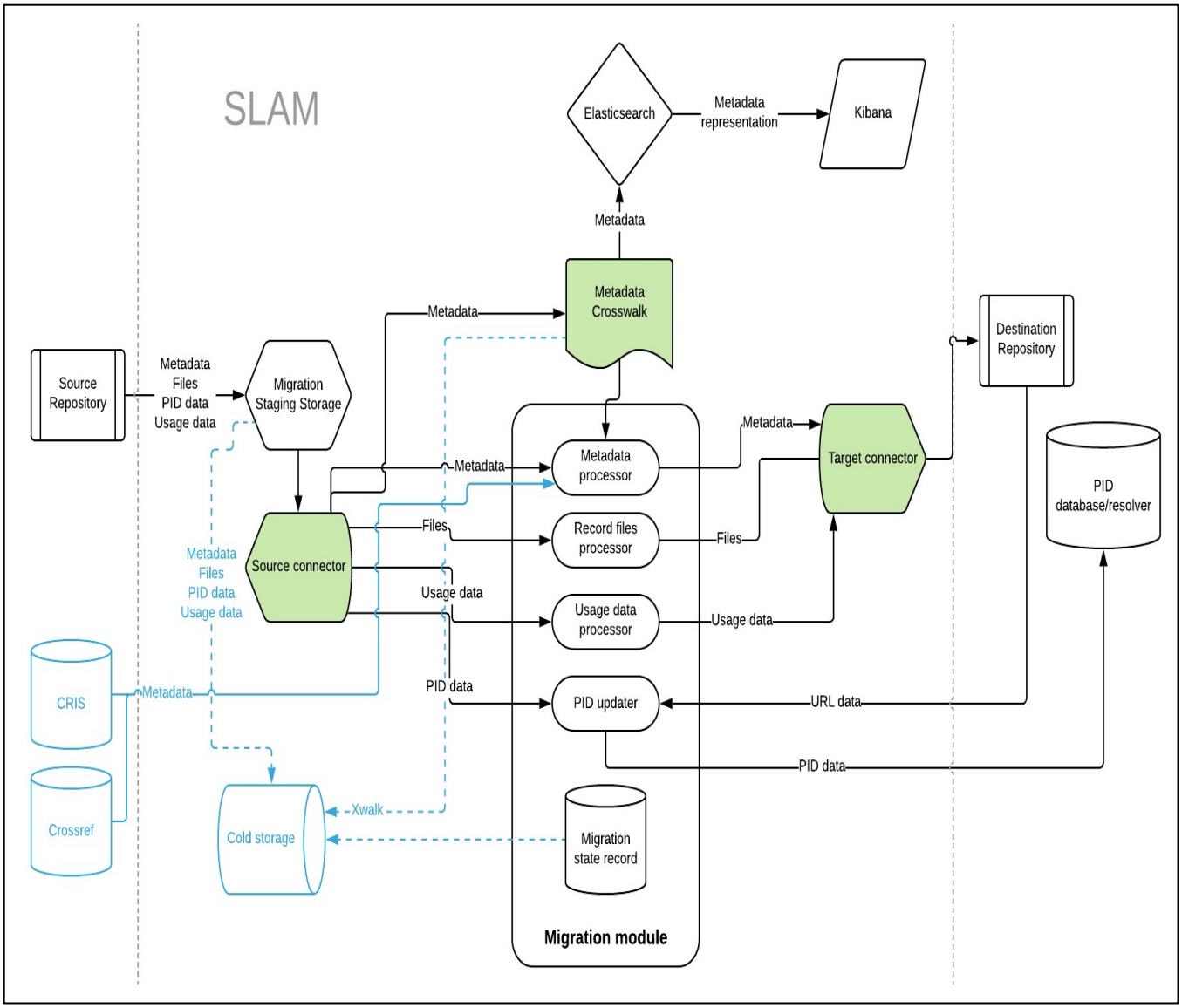

Figure 1. Main components and data flow in SLAM. Areas in light blue are currently under development, while the components highlighted in green need to be adapted for each migration.

The system splits the data to be migrated into four logical slices: bibliographic metadata, record files (e.g., PDFs of journal articles), persistent identifiers of records (PIDs, such as Digital Object Identifiers or handles), and usage data (views and downloads).

Metadata is the first aspect to be considered. From the migration point of view, two dimensions are considered: the syntax and the semantics. Metadata comes in various formats, such as CSV or extensible markup language (XML) files, but most of these can be easily parsed by openly available software solutions. Of more interest are the semantics of the metadata, which stem from the employed *schemas* or ontologies of field definitions; examples include Dublin Core (https://www.dublincore.org) or DataCite (https://schema.datacite.org). A schema crosswalk, which describes how the fields in the target repository schema should be populated using the source data, needs to be set up when transferring records. While this should not be a concern if

the two repositories use the same schema, for the performed migrations (described below) this was not the case. Other reasons for setting up such a crosswalk include

- Loosely defined schema in at least one of the repositories: Certain repository systems do not specify a schema with clear field definitions, validations or applicability. By having the source repository administrators help with setting up a crosswalk, the migration team can avoid issues caused by incomplete understanding of the metadata.
- Support for the review of bibliographic records: Migrations can prove to be an opportunity for reviewing and amending the records' metadata; for example, infrequently used fields can be completely removed, and values which tend to confuse end users can be moved to other fields.
- Ensuring that a record on how the migration was performed, from the metadata point of view, is maintained. The crosswalk is considered an artefact of the migration and is preserved for future reference.

In SLAM, the crosswalk is tested using Elasticsearch, "an open-source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured."[10] The setup uses the crosswalk to create Elasticsearch documents which include all fields as they would be transferred to the destination repository. A Kibana (https://www.elastic.co/products/kibana) dashboard is then used to inspect the records' metadata and perform structured searches across the corpus. This can allow, for example, discovering fields which do not follow a consistent pattern for the values, as seen in figure 2. As the crosswalk includes, apart from the field mapping, altering operations that can be performed on each field, this analysis can facilitate the review process described by the second point above. While performing actual migrations, a number of inconsistencies that the source repository administrators were unaware of were surfaced by SLAM and corrected in the target repository. This is commonplace especially in large corpora spanning decades, where the repository metadata workflows and schemas changed multiple times.

Two points should be noted about this component:

- This is the only component of the architecture for which we mention an actual solution chosen for the practical implementation, namely Elasticsearch. While other solutions could have been chosen, such as the ones included in the Bridge2Hyku toolkit, Elasticsearch proved to be the best fit for a highly automated system which requires analysis capabilities; it is a production-grade solution which can index a high number of documents and support complex queries, while also providing user-friendly analytical views via Kibana.
- There are arguments for loading the metadata in the analysis component without having it processed through the crosswalk; such a workflow could provide further insights into various issues in the corpus which are possibly obscured by the crosswalk. Our practical experiences did not fully justify this requirement, while the actual implementation provided a mean to test the crosswalk, a major migration component; nevertheless, we are still considering the possibility of having to load the raw metadata for analysis in future migrations.

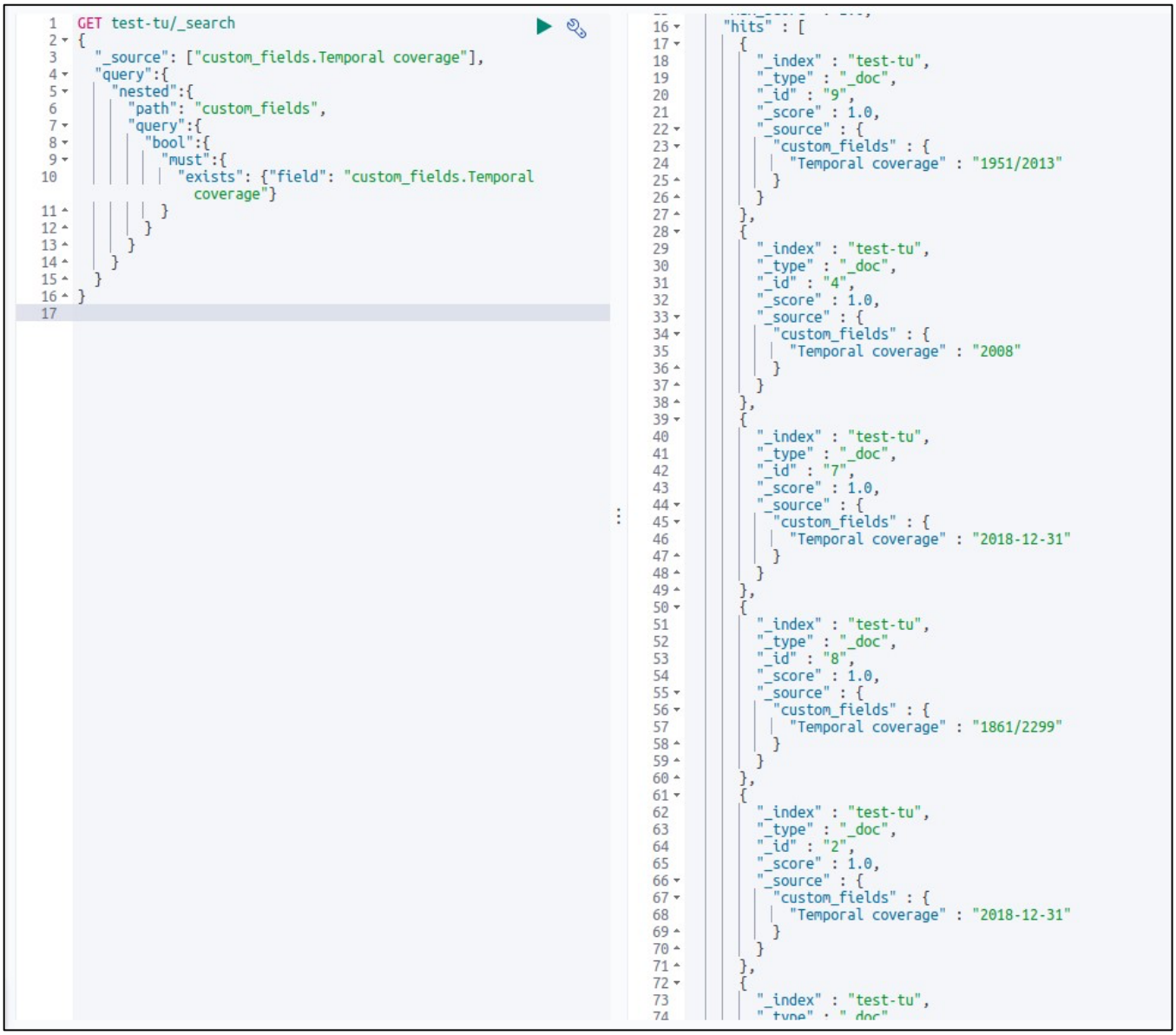

Figure 2. A view examining the possible values of the temporal coverage field from the Dublin Core schema in an institutional repository corpus to be migrated. This shows variation in the format of the values (full date, year only) which can cause issues when migrating to a schema which applies strict validation on date/time values, and thus need to be handled by the migration harness. This view is generated using Kibana from the Elasticsearch stack, employed by SLAM for metadata analysis purposes.

With the crosswalk set up, the migration module can be completed. From a logical point of view, it comprises of four components:

1. Metadata processing: This component uses the crosswalk in order to transfer the metadata to the target repository.
2. File upload: This simply uploads all files associated to a bibliographic record to their new locations.

3. Usage data transfer: Most repositories implement counters for views and downloads of records, and this information, if available, is also transferred to the target repository.
4. Persistent identifier update: If the records are using persistent identifiers, such as Digital Object Identifiers (DOIs) (https://doi.org/) or handles (http://handle.net/), these are updated to resolve to the new locations in the target repository. While employing SLAM for migrations, cases in which persistent identifiers were not employed on source repositories were encountered, with records being accessible only via uniform resource locators (URLs). As these cannot always be transferred across repositories, because each software uses its own URL schema, it is advisable to implement persistent identifiers before migrations.

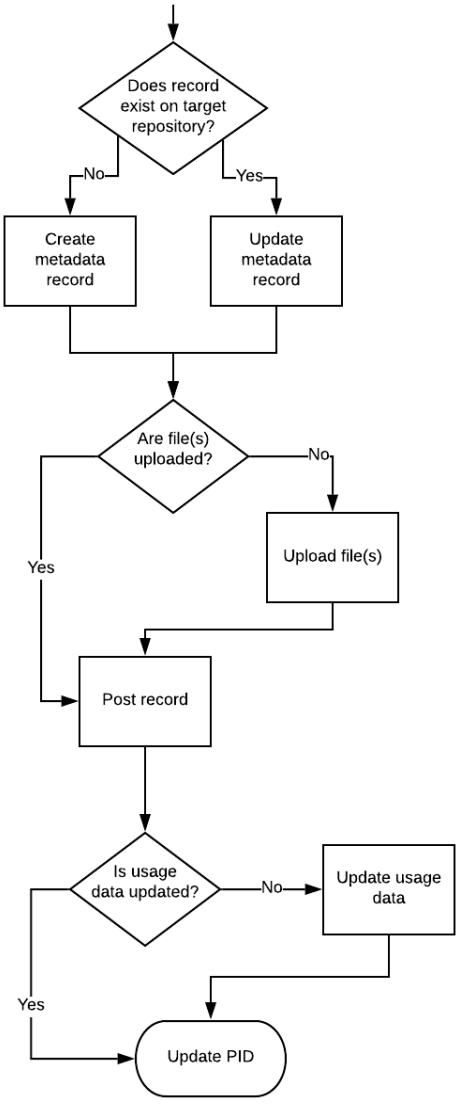

Figure 3. A simplified process diagram describing the steps required for migrating a bibliographic record. Each successful operation is recorded in a persistent database which is used in subsequent runs for resuming the workflow. For example, files will not be uploaded each time the script is run, thus avoiding duplication.

One of the architectural goals of SLAM is statefulness and this is implemented at this level, the migration module being designed as a state machine. A trivial example of such a state machine is shown in figure 3. The state machine status is serialised in a persistent database, with each migration run deserializing it in order to understand which operations still need to be applied for each record. Maintaining such a registry provides several other benefits:

- Facilitates testing and prototyping: This was the original reason behind the architecture, useful especially before the metadata analysis functionality was implemented. If one of the operations required for transferring a record fails, subsequent runs will not apply all steps, but only the ones that did not complete. As for each record a separate state section is maintained, this becomes especially useful when migrating multiple entries; records which failed to migrate can be easily isolated and subsequently reprocessed.
- Allows creating reports on the migration: These are used, for example, to validate that all records were indeed transferred to the target repository.
- Allows the migration module to be portable: If the state machine serialisation is accessible, the module can run from different locations and at different points in time.

The first architectural principle previously presented relates to the reusability of SLAM across migrations. The most common cause of divergence between migrations is related to the differences between repository solutions; SLAM isolates this concern by using two *connectors*, one for the source and one for the target repository. These connectors translate the information to be migrated to and from SLAM's internal data model. Thus, the source connector needs to be able to traverse the staging storage and provide SLAM with all the required record information, while the target connector will upload the records to the new repository (using a web-accessible API for example). This means that for each migration only three parts of SLAM need to be adapted (shown in green highlights in figure 1): the source and target connectors, and the metadata crosswalk. All other components can remain unchanged, thus reducing the technical development time.

In the last step of SLAM's workflow, the information that was used for the migration is sent to a long-term preservation storage, in order to ensure that it remains available for future reference. In our implementation, the following information is preserved:

- Original metadata and files, as extracted from the source repository.
- Metadata crosswalk from source to target repository.
- Migration script state machine serialisation.

This information is sufficient for understanding the exact steps applied during the migration and, if required, for applying certain corrections to the migrated records at a future point in time.

EMPLOYING SLAM FOR REAL-WORLD MIGRATIONS

SLAM was used for performing five repository migrations in one year, as described in table 1; the target repository in all five cases was Figshare.

Table 1. Overview of repositories migrated to Figshare using SLAM.

Source repository identifier	Repository type	Software	Number of records
IR1	Institutional	DSpace	37,000
IR2	Institutional	DSpace	25,605
D1	Data	Custom	334 (105 GB)
IR3	Institutional	Digital Commons	2,275
IR4	Institutional	DSpace	15,474

SLAM's viability was assessed based on the design principles outlined above. Reusability, the main rationale behind SLAM, relates to being able to reuse as much of the system as possible across migrations. The architecture isolated the parts that required adaption from one migration to another (the connectors and the crosswalk); the time spent by a software engineer in order to set up these was monitored. The target here was to support the specialised staff on making domain-specific decisions, especially on the metadata crosswalk, by reducing the time needed to develop the three mentioned components. For example, the Research Excellence Framework (REF) 2021 exercise in the United Kingdom had strict metadata requirements, which required thorough testing in connection with current research information systems and open access monitoring solutions. Between the first and fourth migration, this was reduced from six person-weeks to only two; it is important to note that SLAM evolved between the migrations, based on the lessons learned from each instance.

Statefulness, the property which allows re-processing already-migrated records, is covered in SLAM by the state machine implemented in the migration module, which is persistent and can be referenced in subsequent runs. All the migrations in table 1 required supplementary runs after all records were migrated, most frequently in order to fix metadata issues discovered after the full corpus was transferred. For example, IR1 required three such runs:

1. The first run fixed a number of issues caused by omissions in the metadata schema crosswalk.
2. The second run enriched the metadata using information taken from a current research information system (a source external to SLAM).
3. The last run corrected the usage statistics (view and downloads) which were incorrectly imported initially, due to incomplete understanding of the source repository's database.

Due to SLAM's design, no issues were encountered while performing these runs, as no records were duplicated, removed, or erroneously modified; this was manually checked by the repository administrators, either by sampling the corpus or by inspecting each migrated record, depending on the repository size. A key aspect highlighted by the requirement to reprocess migrated records relates to the granularity of the state machine. As an example, in IR3 a second run required attaching supplementary files to a number of migrated records, and this posed a challenge due to the fact that the state machine only recorded if *all* files have been uploaded, and not *which* files were successfully added to the record. Thus, the state machine was amended to record the complete list of record files, allowing for more granular control over this processing step.

The last concern, fault tolerance, was achieved by applying basic software engineering principles, such as *fail-fast* (report migration issues as soon as they manifest), the implementation of proper exception handling (such as not to ignore any potential issues), and addition of enhanced logging in order to provide a complete record of the processing steps. For each of the five migrations, SLAM ran unsupervised, reporting at the end of each run the records for which an issue was encountered. As an example, in the IR4 migration, SLAM initially failed to migrate 300 records. These were reported to the operator, and after minor fixes were applied to the metadata crosswalk the migration completed successfully. Fault-tolerance plays a central role in ensuring that during migrations no data is lost or corrupted, by surfacing any *edge-case* that might have been missed during the development of the metadata crosswalk, repository connectors, or core migration module, while also isolating such issues to the records exhibiting them, with no impact on the full corpus.

FUTURE DIRECTIONS

While proven viable in real-world scenarios, a number of areas which can benefit from further improvements were identified through an analysis of the current implementation, based on the experiences of the five migrations.

First, the migration-specific components (connectors and metadata crosswalk, shown in green in figure 1) require further decoupling from the core migration module. For example, since all migrations considered Figshare as a target repository, this connector is currently strongly interlinked with the core module, in order to save development time according to business requirements and migration timelines. Further decoupling will ensure that the core migration module's design is not influenced in any way by the repository's architecture and capabilities. Completing this work will also allow making the source code of our current implementation of SLAM publicly available, as in its current state it is making use of proprietary components which are employed across other parts of the Figshare platform. Aside from these, the source code includes straightforward Python modules and makes use of open technologies such as Elasticsearch, which will allow the larger community to adapt and use SLAM with other source or target repositories, or even enhance it with further functionality. Nevertheless, the general architecture can already be implemented in any other way or using a different set of technologies.

Further to this point, the metadata crosswalk is currently influenced by the logic and design of the migration module; for example, it uses the same procedural programming language, Python, as all other components of SLAM. Employing technologies such as eXtensible Stylesheet Language Transformations (XSLT, for metadata in XML formats) or SPARQL (for RDF) will help involve staff with in-depth domain knowledge further in the migration, for whom these technologies are more familiar; moreover, such a design does not require any knowledge of SLAM's internal processes.

Second, the five completed migrations highlighted the importance of reviewing, correcting, and enhancing records during the migration. For example, when migrating a journal article's version of record in an open access context, special care needs to be given to its metadata (title, authors, journal name, publication date or persistent identifier), as mistakes can generate issues with scholarly search engines which will not be able to link the published version to the repository one. A possible input for comparing and correcting existing metadata is the information contained by current research information systems, which aggregate information from various databases, such as Scopus (https://www.scopus.com/). If access to such systems is not available, it is possible to

source metadata from open directories, such as Crossref (https://www.crossref.org/). This component is included in the architectural overview presented in figure 1.

The third area in need of improvement relates to testing the outcome of the migrations. As mentioned in the previous section, this is currently a manual process and can be both cumbersome and error prone. While in line with SLAM's philosophy of automating every step of the process, implementing a mechanism for validating the end migration result could also provide stronger assurances on the completeness and correctness of the migration.

Finally, SLAM's preservation module requires further development in order to ensure that it is fully automated; moreover, the possibility of adding a *manifest* explaining the migration artefacts needs to be considered, as knowledge on the organisation of the information, which is specific to each migration, might be lost in time.

It is important to note that architecture-wise, which was the main concern of this work, we did not identify any major shortcomings in SLAM—most issues discussed above focus on implementation issues. SLAM's modular design will facilitate any additions to the system, required to support new use cases and migrations.

CONCLUSIONS

This paper describes SLAM, the Stateful Library Analysis and Migration system, an ETL software architecture for performing digital library migrations. What differentiates such transfers from other data migrations is the required domain knowledge, the particularities of the target and source repositories in the context of the scholarly communications ecosystem, and the structure of the migration package, which includes, among others, bibliographic metadata, record files, and usage data. Digital libraries are an integral part of the cultural heritage; thus, any migration needs to ensure that no information is lost or corrupted in the process.

The main contributions brought by SLAM are

1. It includes an analysis module based on an industry standard search engine, Elasticsearch, which allows operators to analyse the metadata and schema crosswalk, facilitating the decisions required for properly migrating information between repositories;
2. It implements a serializable state machine in its migration module, which facilitates running the migration procedures multiple times without duplicating, removing, or corrupting records, while allowing for corrections to be applied to the corpus;
3. It follows a modular design, which enhances its reusability across multiple migrations, by reducing the development time required for adapting the system to new source and target repositories.

SLAM applies established software engineering principles in order to provide a trustworthy tool to digital library administrators that need to transfer content between systems. Its design was both influenced and validated by real-world applications, having been used for five different migrations with various requirements and targeted repository solutions.

Future work will consider enhancing SLAM's metadata analysis and enrichment capabilities as well as the collection of further data points on its performance and possible improvement directions while using it for new digital library migrations.

ENDNOTES

[1] David Scherer and Dan Valen, "Balancing Multiple Roles of Repositories: Developing a Comprehensive Repository at Carnegie Mellon University," *Publications* 7, no. 2 (2019), https://doi.org/10.3390/publications7020030.

[2] Directorate-General for Research & Innovation, "H2020 Programme—Guidelines to the Rules on Open Access to Scientific Publications and Open Access to Research Data in Horizon 2020," version 3.2, March 21, 2017, https://web.archive.org/web/20180826235248/http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-pilot-guide_en.pdf; National Institutes of Health, "NIH Public Access Policy Details," last updated March 25, 2016, https://web.archive.org/web/20180421191423/https://publicaccess.nih.gov/policy.htm.

[3] The REF, "Research Excellence Framework," https://web.archive.org/web/20191215143352/https://www.ref.ac.uk/.

[4] Roger C. Schonfeld, "Elsevier Acquires bepress," *Scholarly Kitchen* (blog), August 2, 2017, https://web.archive.org/web/20191212183253/https://scholarlykitchen.sspnet.org/2017/08/02/elsevier-acquires-bepress/.

[5] Alan Hyndman, "Announcing the figshare Institutional Repository… and Data Repository… and Thesis Repository… Really Just an All-in-One Next Gen Repository," *Figshare* (blog), March 22, 2018, https://figshare.com/blog/Announcing_the_figshare_Institutional_Repository_and_Data_Repository_and_Thesis_Repository_really_just_an_all-in-one_next_gen_repository/389.

[6] Alan Hyndman, "Figshare to Power ChemRxiv™ Beta, New Chemistry Preprint Server for the Global Chemistry Community," *Figshare* (blog), August 14, 2017, https://web.archive.org/web/20191218194210/https:/figshare.com/blog/_/322.

[7] bepress, "Digital Commons Dashboard," https://web.archive.org/web/20191218192450/https://www.bepress.com/reference_guide_dc/digital-commons-dashboard/.

[8] Steve Van Tuyl et al., "Are We Still Working on This? A Meta-retrospective of a Digital Repository Migration in the Form of a Classic Greek Tragedy (in Extreme Violation of Aristotelian Unity of Time)," *code{4}lib Journal* no. 41 (August, 9, 2018), https://journal.code4lib.org/articles/13581; Do Van Chau, "Challenges of Metadata Migration in Digital Repository: A Case Study of the Migration of DUO to Dspace at the University of Oslo Library" (master's thesis, University of Oslo, 2011), http://hdl.handle.net/10642/990.

[9] Danijela Tešendić and Danijela Boberić Krstićev, "Business Intelligence in the Service of Libraries," *Information Technology and Libraries* 38, no. 4 (2019), https://doi.org/10.6017/ital.v38i4.10599.

[10] "What Is Elasticsearch?" Elasticsearch BV, http://web.archive.org/web/20191207032247/https://www.elastic.co/what-is/elasticsearch.