Heimdall : retours d'expérience

Régis Witz[1], Guillaume Porte[2], Elsa Van Kote[2], Mohammed Benkhalid[1], Marie Bizais[2], Elisa Michelet[3], Arthur Brody[3], Clément Plancq[4], Titouan Brisset-Sabouraud[5], Bruno Morandière[1], Elsa Ligner[6], and Vincent Jullion[2]

[1] CNRS, France
[2] Université de Strasbourg, France
[3] Bibliothèque Nationale Universitaire de Strasbourg, France
[4] Maison des Sciences de l'Homme Val de Loire, France
[5] Université Libre de Buxelles, Belgique
[6] Contributrice indépendante, France

Abstract

This article describes Heimdall, a mature, streamlined, and highly accessible software solution for scientific data migration and transformation. It highlights some of its features through various real-world use cases related to aggregation, migration, validation, backup, sharing, and publication of research data. It concludes by describing how to add new features, for anyone who wants to save time and avoid developing their own ad hoc migration script.

Mots-clés: logiciel libre, bases de données, interopérabilité, humanités numériques
Keywords: free software, databases, interoperability, digital humanities

1 Introduction

Heimdall est un logiciel permettant de convertir facilement une ou plusieurs bases de données d'un ou plusieurs formats à un ou plusieurs autres. Son implémentation sous forme de librairies Python[1] est documentée et testée de manière approfondie, et est disponible sous licence libre sur la forge logicielle de l'infrastructure de recherche Huma-Num ainsi que sur le repository standard *Python Package Index (PyPI)*. Même si cet article est illustré en langage Python, il existe d'autres portages d'Heimdall : le premier [19] pour l'écosystème BaseX, et un autre [13] dédié à aider les applications web.

Heimdall n'est pas le seul logiciel dans la catégorie *Extract, Transform, Load*[2]. Cependant, il est fait par et pour la recherche publique, et sa création a donc été constamment hantée par les questions suivantes :

— Comment agréger facilement, et de manière reproductible, des données hétérogènes au sein d'un unique corpus ?

Régis Witz, Guillaume Porte, Elsa Van Kote, Mohammed Benkhalid, Marie Bizais, Elisa Michelet, Arthur Brody, Clément Plancq, Titouan Brisset-Sabouraud, Bruno Morandière, Elsa Ligner, and Vincent Jullion. "Heimdall: retours d'expérience." *Actes de la Conférence Humanistica*, éd. par Serena Crespi, Simon Gabay, Martin Grandjean, Ariane Pinche, Marie Puren et Léa Saint-Raymond. Vol. 4. Anthology of Computers et the Humanities. 2026, 10–16. https://doi.org/10.63744/04Kmo7AXXscL.

1. Plus précisément une librairie principale `pyheimdall`, assortie de manière faiblement couplée avec des librairies tierces optionnelles spécialisées, nommées dans cet article «connecteurs». `https://gitlab.huma-num.fr/datasphere/heimdall/connectors/` ; `https://pypi.org/search/?q=pyheimdall`.

2. Il est impossible d'en dresser une liste exhaustive ici, mais Catmandu [12] et Hadoop [6] sont deux exemples d'ETL partagés sous licence libres.

— Comment mettre un corpus au niveau de standards documentaires et de bonnes pratiques qui n'existaient pas quand il a été créé ?
— Comment partager un corpus de différentes manières, pour toucher différents publics, et mieux en pérenniser les données ?
— Comment ne plus perdre autant de temps à gérer la dette technique de vos prédécesseurs ?
— Comment faire tout cela facilement, sans financement, sans ressources humaines, et sans véritable temps libre sur son emploi du temps ?

Plus de détails sur les motivations et les principes sous-jacents d'Heimdall sont disponibles dans l'article de contexte *Interopérabilité des bases de données scientifiques : une sobriété de façade* [20]. Cet article brosse un panorama concret, constitué de retours d'expérience terrain, des différentes situations qu'Heimdall peut aider à solutionner.

2 Exemples d'usages

2.1 Constituer un corpus à partir de sources hétérogènes

Chaque connecteur Heimdall peut permettre de récuperer des données issues d'une source différente, agrégeant ses éléments, entités et propriétés en une nouvelle base de données. Plusieurs bases de données peuvent ensuite être fusionnées ensemble, par exemple afin de permettre des requêtes croisées, ou comparer des éléments plus aisément. Ainsi, un unique corpus peut être constitué, contenant des éléments issus de toutes les sources d'origine et ce, dans un unique format.

```
db_n = heimdall.getDatabase(format='nakala:api', url=...)
db_w = heimdall.getDatabase(format='wikidata:api', url=...)
db_z = heimdall.getDatabase(format='zotero:api', url=...)
db = merge_databases(db_n, db_w, db_z)  # optionnel
```

Créer un nouveau connecteur (comme décrit plus loin) nécessite de connaître le format et les spécificités syntaxiques et sémantiques de la source que ce connecteur gère. Cependant, utiliser ce connecteur (en d'autres termes, écrire les extraits de code cités dans cet article) ne le requiert pas. Ainsi, la personne utilisant Heimdall peut se concentrer sur la logique scientifique de son corpus, sans avoir à apprendre préalablement une syntaxe ou une technologie spécifique… hormis le langage Python, évidemment.

2.2 Publier facilement

Il existe aujourd'hui différents entrepôts institutionnels pérennes, maintenus par différents acteurs publics de la recherche. Citons par exemple Recherche Data Gouv[3], ORTOLANG[4], ou encore Nakala[5].

Considérons uniquement Nakala. Il existe différentes manières d'y publier des données. L'interface graphique de Nakala est utilisable, mais rend relativement chronophage d'y publier des corpus constitués de nombreuses données. Nakala dispose aussi d'une API «REST»[6], malheureusement peu accessible à des utilisateurs ou utilisatrices peu expérimenté·e·s dans l'interrogation de telles API ou peu au fait des comportements spécifiques de celle de Nakala. Enfin, citons Mynkl, une interface web relativement conviviale permettant de publier en lot des données sur Nakala ; cependant, Mynkl attend des données dans un format qui lui est propre[7].

3. `https://recherche.data.gouv.fr`.
4. `https://www.ortolang.fr`.
5. `https://nakala.fr`.
6. `https://api.nakala.fr`.
7. MyNkl est disponible à l'adresse suivante : `https://mynkl.huma-num.fr`. Pour uploader des données sur Nakala avec MyNkl, il faut les recopier dans un modèle de CSV dont la syntaxe (ie. nom et ordre des colonnes) est propre à MyNkl.

Grâce à son connecteur `pyheimdall-nakala` [16], Heimdall constitue une alternative permettant de publier des données en lot sans avoir ni à maîtriser les détails de l'API Nakala, ni à écrire un fichier spécifique, en trois instructions Python[8].

```
# (1) constitution d'une base de données à partir d'un corpus de fichiers locaux
# `url` est un chemin local; `create_item` est une fonction de création d'item
db = heimdall.getDatabase(format='files', url=..., on_item=create_item)
# (2) ajout d'URI standards (par exemple DublinCore)
update_properties(db, ...)
# (3) publication d'une nouvelle collection Nakala
heimdall.createDatabase(db, format='nakala:api', api_key=API_KEY, ...)
```

Le connecteur `pyheimdall-nakala` ne permet pas uniquement de créer des données, mais aussi de les modifier. Cela permet de mettre à jour une collection Nakala existante avec de nouvelles métadonnées. Cette logique de mise à jour simplifiée permet de synchroniser n'importe quelle hiérarchie de dossiers et de fichiers avec Nakala, à la demande ou automatiquement[9].

De plus, enchaîner des appels à `heimdall.createDatabase` avec différents connecteurs[10] permet de synchroniser en une seule fois un même corpus avec plusieurs entrepôts. Cela est moins chronophage, réduit le risque d'erreur humaine, et rend envisageable de nouvelles routes de circulation des savoirs, en grande partie indépendantes des détails d'implémentation des différentes sources de données.

2.3 Partager de manière découplée

Heimdall dispose de nombreux connecteurs, dont certains lisent ou créent de simples fichiers texte tels que XML, JSON, YAML ou CSV. Combiné aux fonctionnalités de filtrage et de tri d'Heimdall, cela permet de partager aisément une « vue » partielle d'un corpus plus large pour, par exemple, en exclure les données sous embargo, ou non validées scientifiquement.

```
# (1) importe une base SQL, un ne prenant que certaines tables
db = heimdall.getDatabase(
  format='sql:mariadb',
  url=f'mariadb://{username}:{password}@{host}:{port}/{database}',
  entities=('only', 'those', 'tables'),
)
# (2) (optionnel) filtre encore davantage les données
moar_database_filtering(db)
# (3) partage uniquement les données filtrées, ici avec un SQL dump et un XML}
heimdall.createDatabase(db, format='sql:mariadb', url='filtered_dump.sql')
heimdall.createDatabase(db, format='hera:xml', url='filtered_data.xml')
```

Ainsi, Heimdall permet de décorréler les logiques de stockage, de partage et de visualisation des données[11].

8. C'est ce que fait le projet *CiSaMe* [8], qui a publié ses données sur Nakala grâce à Heimdall, simplifiant grandement le code à écrire et à maintenir, puisque celui-ci a pu se concentrer sur sa logique projet. Le résultat [4], dont cet article donne une version simplifiée, représente moins de 100 lignes de pur alignement des données, contre les plusieurs milliers de lignes nécessaires à implémenter un tel uploader *ad hoc* sans utiliser Heimdall.

9. Ce besoin de mise à jour d'une collection Nakala existante a été exprimé par la plateforme *ESTRADES* [10] pour le projet *HistCarto* [7]. La fonctionnalité correspondante a été rajoutée à `pyheimdall-nakala`, avec la même logique d'abstraction et de mutualisation incrémentale que pour *CiSaMe*7.

10. Par exemple `pyheimdall-nakala`, déjà abordé dans cet article, ou encore `pyheimdall-sword` [18], qui permet d'agréger des données issues de différents serveurs supportant le protocole SWORD [14], tels qu'arXiv ou une instance de Dataverse telle que Recherche Data Gouv3.

11. C'est ce que fait le projet *Chinese Knowledge in Poetry* [3], qui partage d'une part les données filtrées ainsi que les scripts de filtrage eux-mêmes [1], et d'autre part ses outils de visualisation [2], des Notebooks Jupyter utilisant à leur tour Heimdall pour en simplifier la lecture et en améliorer les performances.

2.4 Migrer d'un système à l'autre

De nombreux systèmes de gestion de bases de données sont aujourd'hui à la disposition des chercheuses et des chercheurs [12]. Leur durée de vie, et la confiance qu'on peut leur accorder (en termes de maturité, de sécurité, de transparence de la gestion de projet, etc) sont variables. Pour préserver, réutiliser ou libérer des données scientifiques, il est donc parfois nécessaire de migrer d'un système à l'autre.

Un des objectifs de départ d'Heimdall est de rendre de telles migrations moins douleureuses, chronophages et risquées. Cette section illustre concrètement qu'il suffit généralement de quelques instructions Python.

2.4.1 Libérer des données historiques

Pour le musée Adolf Michaelis [13], Heimdall a permis de récupérer les données d'un serveur CollectiveAccess qui allait être fermé, puis de les exporter au format CSV, afin de pouvoir lancer un projet étudiant visant à les mettre en valeur via un site web et une application tablette.

```
db = heimdall.getDatabase(format='collectiveaccess:api', url=...)
heimdall.createDatabase(db, format='csv', url=..)
```

2.4.2 Préserver des données

Heimdall permet de convertir automatiquement toute base de données en une véritable base SQL, avec un schéma complet et directement utilisable [14]. Par exemple, toutes les bases d'une équipe anciennement dans Heurist ont pu être transformées ainsi, puis exploitées de manière maîtrisée, sans plus avoir à craindre les régressions et indisponibilités de ce logiciel SaaS [15]. L'export vers différents formats texte, hébergés ensuite sur un serveur de partage de fichiers institutionnel, a aussi été une réassurance.

```
db = heimdall.getDatabase(format='heurist:api', url=...)
heimdall.createDatabase(db, format='sql:mariadb', url=...)  # à sourcer par votre serveur SQL
heimdall.createDatabase(db, format='hera:json', url=...)  # à stocker sur ShareDocs, par exemple
```

2.4.3 Parler le web sémantique

Heimdall encourage et facilite le fait de documenter scientifiquement une base relationnelle en y ajoutant des descriptions et des URI standards, même si la technologie d'origine des données ne le permet pas. Comme décrit précédemment, cette phase de documentation, d'alignement avec des vocabulaires scientifiques contrôlés, est aujourd'hui obligatoire pour publier ses données sur un entrepôt véritablement pérenne. Une fois que cette documentation est faite, exporter en RDF est trivial.

```
db = heimdall.getDatabase(format='sql:mariadb', url=...)
update_properties(db, {  # ajout d'URI standards
  'name': [
    'http://purl.org/dc/terms/title',
    'http://xmlns.com/foaf/0.1/name',
    ],
```

12. Un exemple de liste des systèmes de bases de données existants : `https://db-engines.com/en/ranking/` . Cette liste contient plus de 400 éléments, mais est malgré tout encore loin d'être exhaustive. Par exemple, elle oublie la plupart des tableurs, et des logiciels accessibles en SaaS.
13. Site de l'Association des Amis du Musée Adolf Michaelis : `https://amamstrasbourg.org/`
14. Par «directement utilisable», dans un contexte de sortie d'Heurist, j'entends deux choses. Premièrement, du SQL en mode texte, directement importable quel que soit le moteur utilisé, et pas du binaire MySQL qu'il faudra certainement convertir techniquement derrière, car MySQL est visiblement en perte de vitesse. Deuxièmement, une table par *Record type*, et pas le schéma interne d'Heurist qu'il est nécessaire d'avoir étudié en amont, et qu'il faudra convertir logiquement derrière.
15. `https://heurist.huma-num.fr`.

```
  'author': [
    'http://purl.org/vocab/frbr/core#creator',
    ],
    ...
 })
heimdall.createDatabase(db, format='rdf:turtle', url=...)
```

L'exemple ci-dessus montre un export au format RDF Turtle. Cependant, les formats N-Triples, Hextuples, TriG, XML et JSON-LD sont aussi supportés par le connecteur `pyheimdall-rdf` [17].

2.4.4 Valider l'intégrité des données

Il existe de nombreuses manières de corrompre les données lors de leur migration [5]. Puisque tous les imports, exports et diverses modifications que permet Heimdall passent par la même représentation interne, il est aisé de déclencher systématiquement les mêmes procédures qualité lors de chaque migration de données, quels qu'en soient les formats d'origine et de destination.

```
db = heimdall.getDatabase(format=input_format, url=...)
assert is_good_quality(db)
heimdall.createDatabase(format=output_format, url=...)
db_after = heimdall.getDatabase(format=output_format, url=...)
assert is_good_quality(db_after)
assert is_same(db, db_after)
```

Heimdall lui-même est automatiquement et rigoureusement testé, que ce soit fonctionnellement ou unitairement, à chaque fois qu'il est modifié. Cela ne garantit pas qu'il soit dénué de défaut [9], mais cela garantit la cohérence de tous les comportements qui sont testés. Comme le nombre de tests croît naturellement avec le temps en fonction des retours terrain, la maturité du logiciel croît en même temps.

2.5 Créer son propre connecteur

Un connecteur étant une extension (*plugin*) d'Heimdall, n'importe qui peut déclarer le sien. Ce mécanisme est le moins intrusif possible : il ne nécessite que de créer une fonction dotée d'un décorateur spécifique [11].

```
import heimdall

# connecteur d'import de données
@heimdall.decorators.get_database('exemple')
def getDatabase(**options):
    db = heimdall.util.tree.create_empty_tree()
    # TODO: peupler `db` avec des données importées
    return db

# connecteur d'export des données
@heimdall.decorators.create_database('exemple')
def createDatabase(db, **options):
    pass # TODO: exporter les données de `db`
```

Le profil de cette fonction décorée est le plus flexible possible. De plus, du moment qu'Heimdall est disponible dans l'environnement Python, chaque connecteur déclaré est automatiquement détecté et utilisable avec toutes les fonctions d'Heimdall. Cela permet à chacun de gérer l'implémentation et le déploiement de son propre connecteur de manière cohérente avec ses propres critères de qualité, qui peuvent différer de ceux des contributeurs et contributrices d'Heimdall.

Pour simplifier encore le développement et le déploiement de nouveaux connecteurs, il existe un dépôt d'exemple [15].

3 Conclusion

Heimdall a aujourd'hui atteint une forme de stabilité. Sa qualité croît naturellement, en ne nécessitant aucune maintenance ou presque. Pour exister sur la durée, Heimdall ne requiert donc ni financement, ni communauté.

L'ajout de nouveaux connecteurs se fait au gré des besoins, par n'importe qui, sans impacter ni Heimdall, ni les autres connecteurs. Développer ces connecteurs au fil de l'eau, puis les partager publiquement, est une forme de mutualisation et améliore la transparence et la reproductibilité de la recherche.

Références

[1] Bizais-Lillig, Marie. « CHI-KNOW-PO : Knowledge base ». 2024. URL : https://gitlab.huma-num.fr/chi-know-po/knowledge-base.

[2] Bizais-Lillig, Marie. « CHI-KNOW-PO : Online tools ». 2024. URL : https://gitlab.huma-num.fr/chi-know-po/tools.

[3] Bizais-Lillig, Marie. « Chinese Knowledge in Poetry ». 2017. URL : https://chi-know-po.gitpages.huma-num.fr.

[4] Brisset-Sabouraud, Titouan et Witz, Régis. « Uploader Nakala pour CiSaMe ». 2025. URL : https://gitlab.huma-num.fr/estrades/projets/cisame/nakala-uploader.

[5] Casties, Robert. « There and Back Again—How to Preserve Your Data During Transformations ». In : *Digital Humanities Tech Symposium 2025*, sous la dir. de Julia Damerow et Rebecca Sutton Koeser. Anthology of Computers et the Humanities, 2025, p. 6-12. DOI : 10.63744/EZFV8PwG9Frz.

[6] Cutting, Doug et Cafarella, Mike. « Apache Hadoop ». 2006. URL : https://hadoop.apache.org.

[7] Laboulais, Isabelle. *Les Usages des cartes (XVIIe-XIXe siècle) – Pour une approche pragmatique des productions cartographiques*. Strasbourg : Presses Universitaires de Strasbourg, 2008. DOI : 10.4000/books.pus.13387. URL : https://histcarto.u-strasbg.fr.

[8] Leveleux-Teixeira, Corinne et Eckert, Raphaël. « The CiSaMe Project – Circulation of medieval knowledge in the 12th century ». In : *17th Congress of Medieval Canon Law*. https://anr.hal.science/hal-04889792. Canterbury, Angleterre, 2024. URL : https://cisame.hypotheses.org/.

[9] Nyman, Jeff. « The Zero Defect Fallacy ». 2019. URL : https://testerstories.com/2019/10/the-zero-defect-fallacy.

[10] Porte, Guillaume. « Estrades (Solutions de transcription, d'analyse et de diffusion pour l'édition structurée) ». 2024. URL : https://estrades.huma-num.fr.

[11] Smith, Kevin D., Jewett, Jim J., Montanaro, Skip et Baxter, Anthony. « PEP 318 – Decorators for Functions and Methods ». PEP 318. Python Enhancement Proposals (PEPs), 2003. URL : https://peps.python.org/pep-0318.

[12] Steenlant, Nicolas et al. « Catmandu ». https://librecat.org. 2016. URL : https://metacpan.org/pod/Catmandu.

[13] The Heimdall.js contributors. « Heimdall.js ». 2026. DOI : 10.5281/zenodo.10671999.

[14] THE PRESIDENT & FELLOWS OF HARVARD COLLEGE. « SWORD API ». Dataverse Guide. Institute for Quantitative Social Science, 2025. URL : https://guides.dataverse.org/en/latest/api/sword.html.

[15] THE PYHEIMDALL CONTRIBUTORS. « Heimdall – Example Connector ». 2024. URL : https://gitlab.huma-num.fr/datasphere/heimdall/connectors/example.

[16] THE PYHEIMDALL CONTRIBUTORS. « pyheimdall-nakala ». 2025. URL : https://pypi.org/project/pyheimdall-nakala.

[17] THE PYHEIMDALL CONTRIBUTORS. « pyheimdall-rdf ». 2025. URL : https://pypi.org/project/pyheimdall-rdf.

[18] THE PYHEIMDALL CONTRIBUTORS. « pyheimdall-sword ». 2026. URL : https://pypi.org/project/pyheimdall-sword.

[19] THE XHEIMDALL CONTRIBUTORS. « xHeimdall ». 2024. DOI : 10.5281/zenodo.10665319.

[20] WITZ, Régis et PORTE, Guillaume. « Interopérabilité des bases de données scientifiques : une sobriété de façade ». In : *Humanités numériques 11* (2025). DOI : 10.4000/1498w.