Vol.2020-IOT-51 No.11
Vol.2020-SPT-39 No.11
2020/9/4

研究再現性を支える情報基盤が持つべきデータモデルの検討

藤原一毅 [1]　常川真央 [1]　合田憲人 [1]　山地一禎 [1]

概要：オープンサイエンスの普及にともない、公開された研究成果を第三者が容易に再現・再利用できるシステムが各所で開発されている。一口に「研究成果の再現・再利用」と言っても、対象ユーザー（汎用的なもの／特定の研究分野に特化したもの）や再現すべき事物（データ来歴を保証する／計算精度を保証する／etc）の点で各システムは性格を異にし、それぞれに特化したデータモデルを持っている。国立情報学研究所 (NII) では、研究データ管理サービス NII Research Data Cloud (RDC) の一環として、研究成果の再現・再利用をサポートするデータ解析サービスを構想している。本サービスは、研究データ管理計画やデータリポジトリなどの関連システムと統合された一体的なユーザー体験の提供を目指している。その中で、関連システムとの連携にどのようなデータモデルを用いるべきかは、データ解析サービスが何を／誰を対象とするのかとも密接に関わる問題であり、サービスの将来像を踏まえて俯瞰的に検討するべき課題である。本稿では、研究再現性をサポートする既存システムの設計をサーベイするとともに、NII RDC データ解析サービスが持つべきデータモデルに関する検討内容を報告する。

キーワード：研究再現性, 研究データ管理, オープンサイエンス

Investigating data models for research reproducibility platforms

Ikki Fujiwara[†1]　Mao Tsunekawa[†1]　Kento Aida[†1]　Kazutsuna Yamaji[†1]

1. はじめに

実験科学の文脈で再現性の危機が叫ばれて久しい[1]。データ中心科学が普及した今日では、計算科学の文脈でも同様に、再現性の担保が研究の信頼性を左右する重要な問題として認識されている[2]。日本では、公的研究資金による研究成果のうち、論文及び論文のエビデンスとしての研究データについては原則公開とし、その他研究開発成果としての研究データについても可能な範囲で公開することが望ましいとする指針が示された[3]。しかし、単にデータを公開するだけでは研究の再現性や再利用性は保証されない。そのデータを元に科学的発見を得るに至ったプロセスを公開し、発表後に他の研究者が同じプロセスを追跡できるようにすることで、初めて研究の再現性が保証される[4]。したがって、データと同等の重要性がプログラム[a]とその実行環境にも認められなければならない[5]。

研究活動のライフサイクルの中で解析結果の再現性が求められる場面は以下の 4 つに分けられる。

（ア）**発表前：共同研究のための再現性**　研究者は、他の研究者（同僚や学生を含む）と協力してデータ解析を行う場合、実際の作業者にかかわらず一貫した結果を得なければならない。そのためには、プログラムとその実行環境を共通化し、作業手順を標準化することが必要である[6]。

（イ）**論文投稿時：査読のための再現性**　いくつかのジャーナルや国際会議では、論文に掲載された図表などを査読者が検証できるよう、必要なデータとプログラムを著者が提出しなければならない[7][8]。この場合、著者は、自身の手元で動かしていたプログラムを第三者である査読者が動かせる形に取りまとめて提供することが求められる[9]。

（ウ）**発表後：派生研究のための再現性**　ある研究者が、他の研究者の成果を出発点として自らの研究を開始するために、元の研究に使われたデータとプログラムを入手し、再利用する。この場合、入手した研究者がプログラムを正常に動かすためには元と全く同じ実行環境を再構築することは容易ではない[10]。

（エ）**不正発生時：遡及のための再現性**　いくつかの国や学術機関は、研究不正発生時の調査を可能とするために、研究成果の根拠となるデータの保存を研究者や研究機関に義務付けている[11]。この場合、来歴の長期保存と改ざん防止が課題となる[12]。

このように、研究活動の各場面で異なるステークホルダーが異なる目的をもって再現可能化を要請していることが、しばしば研究再現性に関する議論を複雑にしている。

近年、研究成果を容易に再現可能化できると謳うツールやプラットフォームが次々に出現している。一方、国立情報学研究所（NII）では、データ駆動型時代の研究を支援するプラットフォームである NII Research Data Cloud（RDC）の開発を進めている。NII RDC は、オープンサイエンスの理念に基づき、分野の垣根を超えた研究の発展を促進するため、分野中立的な研究再現性プラットフォームとなることを構想している。すなわち、コンピュータに詳しくない研究者がデータとプログラムを含む研究成果を NII RDC に

1 国立情報学研究所
National Institute of Informatics

a 本稿では「ソフトウェア」「プログラム」「コード」を同じ意味で使う。

公開し、他分野の研究者がその成果を NII RDC で容易に再現できることを目指す。これを実現するには、プログラムとその実行環境を論文や研究データと同格の実体として扱うための適切なデータモデルを設計する必要がある。そのための予備調査として、本稿では研究再現性をサポートする既存のツールやプラットフォームをサーベイした。

以下、第 2 章で研究再現性に関わる 3 つの論点を整理した後、第 3 章で既存システムの調査結果を報告する。最後に第 4 章で、各システムの設計を比較し、分野中立的な研究再現プラットフォームに適した設計を考察する。

用語

研究再現性に関する議論ではしばしば用語の混乱が見られる。本稿では ACM の定義[13]を採用する。すなわち、計算科学の文脈において

- Repeatability（繰返し性）とは、同じチームが同じデータ・同じ条件（プログラムや実行環境）で繰り返し実験を行い、毎回同じ結果が得られること
- Reproducibility（再現性）とは、異なるチームが同じデータ・同じ条件で実験を行い、元の実験と同じ結果が得られること
- Replicability（複製可能性）とは、異なるチームが異なるデータ・異なる条件で実験を行い、元の実験と同じ科学的結論が得られること

を意味する。この定義は NASEM Report の定義[14]と矛盾しない一方、Biointerphases 誌の定義[15]とは矛盾する。用語に関する議論は文献[16]に詳しい。

2. 論点整理

本稿では、計算科学において再現性が損なわれる原因を以下の 3 つの論点によって整理する。すなわち、誰が実験するか (who)、どこで実験するか (where)、いつ実験するか (when) の 3 軸である。

2.1 どこで実験するか（依存性の問題）

プログラムの実行環境を全く同一にすることの難しさは、実験の再現性を損なう大きな要因である。ここでいう実行環境とは、ハードウェアだけでなく、実験プログラムを実行するために必要なソフトウェアスタック全体を含む。GPU と CPU で計算結果が違ったり、ライブラリのバージョン違いによってエラーが発生したりする問題は、すべて依存性の問題に含まれる。仮想マシン、コンテナ、エミュレータといった仮想化技術は依存性の問題を回避する方策である。並列計算の精度や数値計算の安定性の問題も「どこで実験するか」という論点に含まれるものと考える。

2.2 誰が実験するか（属人性の問題）

人づてのデータ入手、GUI を用いたファイル管理、スプレッドシートを用いた前処理、手打ちコマンド内でのパラメータ指定など、研究者が持つ暗黙知が実験手順に含まれる場合、実験の再現性が損なわれる。Jupyter Notebook [17] のような文芸的コンピューティングシステムは、ソースコードと自然言語による説明を強く関連付けて記述することで属人性を緩和しようとする方策と考えられる。しかし、たとえ暗黙知がドキュメント化されていたとしても、ヒューマンエラーがひとつでも混入すれば実験は再現不能となりうる。データ来歴を自動的に記録するシステムは、ヒューマンエラーを排除し再現性を高めるひとつの手段である。

2.3 いつ実験するか（永続性の問題）

研究成果が公表されてからその再現が求められるまでの間には数年～数十年のタイムラグが存在しうる。タイムラグが小さければ仮想化によって古い実験プログラムを実行できる可能性があるが、タイムラグが大きい場合、ハードウェアやソフトウェアの寿命といった技術的要因だけでなく、人間や組織の寿命といった社会的要因によっても再現性が損なわれる。例えば、実験データが保存された CD-R が 10 年後に読み取り不能となったり、実験プログラムが依存しているプロプライエタリソフトウェアが 20 年後に入手不能となったり、暗黙知を持つ人材が 30 年後に死亡したりすることが容易に想像できる。研究成果の再現可能化を謳うプラットフォームを検討するにあたっては、そのプラットフォームがどの程度の永続性を想定しているのかを見極めなければならない。

近年次々に出現している研究再現プラットフォームはいずれも、上述した 3 つの問題のうち 1 つ以上に対処しようとするものである。次章では、これら 3 つの観点から各プラットフォームの特徴を整理する。

3. サーベイ

3.1 Jupyter(Hub), Binder(Hub)

Jupyter Notebook [17]は、プログラムとその実行結果をドキュメントとともに保存するオープンなフォーマットと、それを実行するウェブアプリケーションである。コードとドキュメントを一体化することで属人性の問題を緩和する効果が期待され、単体でもシンプルな電子実験ノートとして広く利用されるほか、以下に挙げる多くのプラットフォームでフロントエンドとして採用されている。JupyterHub は複数のユーザーに Jupyter を提供する。

Binder [18]は、Git などのリポジトリに保存された環境構成情報（environment.yml や Dockerfile など）に基づいて、依存関係が解決された実行環境（コンテナ）を自動的に再構築する。BinderHub は Kubernetes 上で複数のユーザーに Binder を提供する。Binder は内部で Conda や Pip など処理系のパッケージ管理システムを呼び出し、作成者による記述に従ってパッケージをインストールする。これにより、再現者にとって依存性の問題が緩和される。作成者には、自らの実行環境の再現に必要なパッケージを把握して環境構成情報を記述できるスキルが要求される。Binder で再構築可能なリポジトリが持つデータモデルを図 1 に示す。

Binder は永続性の問題を解決しない。もしパッケージリポジトリやコンテナリポジトリが廃止されたら、そこにあるパッケージやイメージに依存する実行環境を再構築できなくなる。

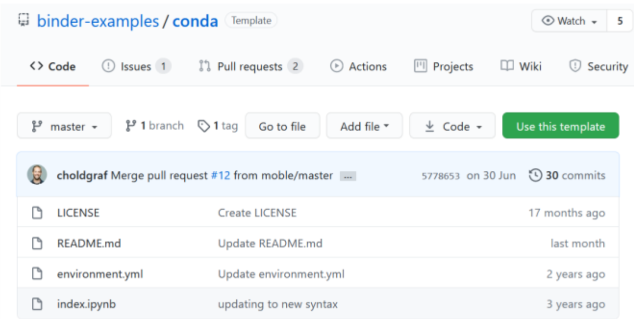

図 1　Binder 対応リポジトリ

3.2　Code Ocean

Code Ocean [19]は、研究再現性に関わるサービスを統合的に提供するフリーミアムな PaaS である。ソースコード、データ、環境構成情報（Dockerfile）およびメタデータを含む「計算カプセル」(compute capsule) をユーザーが作成し、Code Ocean 社のクラウド上で実行、保存、公開することができる。計算カプセルの実体は標準化されたディレクトリ構造をもつ Docker コンテナである。

ユーザーは、はじめにベースとなるコンテナイメージと追加パッケージを GUI で指定して計算カプセルを作成する。次に、計算カプセル内にソースコードとデータをアップロードし、必要なメタデータを記述して、プログラムを実行する。最後に「Submit for publication」ボタンを押すと、同社のスタッフが再現性を確認し、問題なければ DOI が付与されて同社の公開リポジトリに掲載される。掲載された計算カプセルは、他のユーザーが発見、複製、再利用できる。計算カプセルが持つデータモデルを図 2 に示す。

Code Ocean の特長は、公開データストレージ、ソースコードリポジトリ、コンテナレジストリ、計算カプセルのリポジトリと検索サービスなど、コンテナベースの再現可能化に必要なサービス群を同社がワンストップで提供している点にある。Docker をはじめとするオープンソースソフトウェアの組み合わせでも同様のことが実現できるとはいえ、コンピュータの専門家でない研究者にとって、Code Ocean が提供する統一的な UX と有人サポートは、属人性と依存性の緩和に役立つと思われる。

Code Ocean は永続性の問題を解決しない。もし同社が倒産したら、クラウド上に保存された計算カプセルは消滅するだろう。ただし、計算カプセルを ZIP ファイルとしてエクスポートし、ローカルの Docker 上で再利用することは可能である。

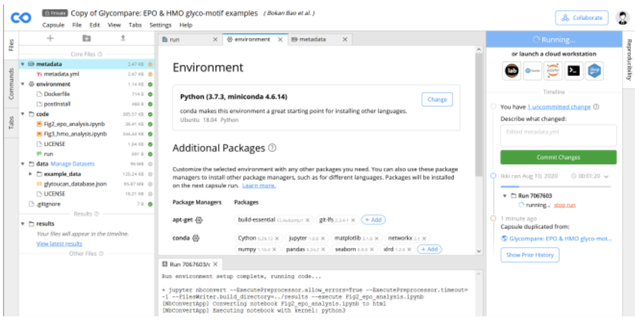

図 2　Code Ocean の実行画面。左ペインにデータが見える

3.3　The Whole Tale

The Whole Tale [20]は、研究再現性をサポートするオープンソースの PaaS である。ソースコード、データ、環境構成情報およびメタデータを含む「テール」(Tale) をユーザーが作成し、wholetale.org のクラウド上で実行できる。The Whole Tale はデータとテールの公開リポジトリを内部に持たず、外部のデータリポジトリに依存している点が Code Ocean と異なる。

ユーザーは、はじめに実行環境（Jupyter や RStudio など）を指定してテールを作成する。次に、データとソースコードをテールに追加する。The Whole Tale には①ホームディレクトリ（すべてのテールが読み書き可）、②ワークスペース（そのテールのみが読み書き可）、③外部データ（すべてのテールが読み取り可）の 3 種類のデータ領域があり、所定の外部リポジトリ（DataONE, Dataverse, Globus）にあるデータセットは③に登録することができる。この場合、FUSE を介してアクセスした時点で初めて実体が転送され、内部にキャッシュされる。データとソースコードが揃ったら、テール内の実行環境を用いてプログラムを作成・実行する。実験が完了したら、必要なメタデータを記述し、実行結果を含むテールを外部リポジトリ（DataONE または Zenodo）に直接公開できる。DOI は外部リポジトリ側で付与される。他のユーザーは外部リポジトリ上でテールを発見し、wholetale.org のクラウド上に複製、再利用できる。テールが持つデータモデルを図 3 に示す。

The Whole Tale は Code Ocean と同様に属人性と依存性の問題を緩和する。作成者には、用意された実行環境に足りないパッケージを把握して環境構成情報を記述するスキルと、適切な外部リポジトリを選択するスキルが要求される。

The Whole Tale はテールの保存・公開を外部リポジトリに委譲することで永続性の問題を回避している。同プロジェクトが終了したらクラウド上の実行環境は廃止されるだろう。ただし、テールをエクスポートしてローカルの Docker 上で再利用することは可能である。

```
format: 3
metadata:
  name: Humans and Hydrology Test
  identifier: '8e475f85-d7af-465f-97a1-198b9acdc4fb'
  authors:
    - name: Craig Willis
      orcid: https://orcid.org/0000-0002-6148-7196
  category: science
  description: Test of tale serialization format
  illustration: https://raw.githubusercontent.com/whole-tale/.../demo-graph2.jpg
  entrypoint: wt_quickstart.ipynb
  public: true
data:
  - source: DataONE
    url: http://cn.dataone.org/cn/v2/resolve/urn:uuid:1d23e155-3ef5-47c6-9612-027c80855e8d
  - source: HTTP
    url: http://example.com/data.csv
files:
  - path: notebooks/wt_quickstart.ipynb
    url: https://cn.dataone.org/cn/v2/resolve/urn:uuid:71359f62-b260-4793-a866-418f7fa73aaa
  - path: environment/docker-environment.tar.gz
    url: https://cn.dataone.org/cn/v2/resolve/urn:uuid:71359f62-b260-4793-a866-418f7fa73aaa
environment:
  name: Jupyter Notebook
  url: https://github.com/whole-tale/jupyter-yt
  commit: dc91deafdc959c7edcb8199171b5ac75763323e
  icon: https://raw.githubusercontent.com/whole-tale/rstudio-base/master/RStudio-Ball.png
  archive: environment/docker-environment.tar.gz
  config:
    - command: /init
      environment: CSP_HOSTS=dashboard.dev.wholetale.org,
      port: 8787
      targetMount: /home/rstudio/work
      user: rstudio
```


図 3　The Whole Tale のテールのデータモデル[21]

3.4　KBase

KBase [22]は、バイオインフォマティクス分野で微生物や植物のデータ共有と解析を行うオープンソースの PaaS である。Jupyter Notebook を拡張した「ナラティブ」と呼ばれるノートブックを用いて複数のデータとアプリを含むワークフローを作成し、KBase のクラウド上で実行、保存、共有、公開することができる。

ユーザーは、はじめにデータをナラティブに登録する。外部のデータリポジトリから KBase にミラーされている微生物ゲノム、植物ゲノム、培地組成、反応、化合物などのほか、共同研究者から共有されたデータや自らアップロードしたデータも利用できる。次に、アプリを KBase のアプリカタログから選んで登録する。アプリには入出力データ型[23]が定義されていて、登録済みのデータに対応するアプリを絞り込むことができる。登録したアプリに入力データとパラメータを設定して実行すると、生成された出力データが自動的にナラティブに登録される。この出力データを別のアプリに入力することで、ユーザーは複雑なワークフローをナラティブとして作成する。ユーザーは、ナラティブを他のユーザーと共有して共同編集したり、KBase のライブラリに公開したりできる。

KBase はデータの来歴を保持しており、生成過程を検証・再現可能である。これにより、属人性の問題を解決する。来歴の実体は、バックエンドにおいて「ワークスペース」と呼ばれる型付きオブジェクトのコレクションとして実装される[24]。各オブジェクトのインスタンス（フロントエンドにおけるデータやアプリに対応する）は自動的にバージョン管理され、その生成過程を記述した来歴情報にリンクされている。ナラティブがコピーされると、そのナラティブに登録されているオブジェクトもコピーされる。KBase のアプリは KBase SDK を用いてラップされた（多くは既存の）バイオインフォマティクスツールであり、各々が Docker コンテナ内で実行される[25]。これにより、依存性の問題が解決される。

KBase は永続性の問題に言及していない。

3.5　AiiDA

AiiDA [26]は、主に計算材料科学分野向けの、再現可能なワークフローを自動実行するオープンソースのプラットフォームである。AiiDA 本体はスタンドアロンで利用するツールであり、AiiDA コア、データベース、ワークフローエンジンからなる。ユーザーは、AiiDA の Python API やプラグインを用いてプログラムを書く。AiiDA の CLI を介してこのプログラムを実行すると、データとコードを含む計算グラフが自動的に抽出され、来歴としてローカルのデータベースに保存される。来歴のデータモデルを図 4 に示す。プログラムの実行はワークフローエンジンによって管理され、バッチスケジューラを介して外部の HPC を利用できる。再利用可能なプログラムを作成したら、プラグインとして AiiDA プラグインレジストリに公開できる。AiiDA lab [27]は、Jupyter Notebook 上で AiiDA を利用できる PaaS である。

Materials Cloud [28]は、主に AiiDA の来歴を収録する公開リポジトリである。実験が完成したら、ユーザーは来歴をエクスポートし、関連論文などのメタデータを添えて投稿する。すると、モデレータが来歴の内容を確認し、問題なければ DOI が付与されて Materials Cloud Archive [29]に掲載される。

AiiDA は、AiiDA API を用いて書かれたプログラムによる計算来歴を自動的に記録する。これにより、属人性の問題を解決する。一方で、AiiDA は依存性の問題を解決しない。来歴には入出力データとプラグイン名に加えて計算機の情報（ホスト名やスケジューラのオプションなど）が記録されるが、計算環境を再現する機能は含まれていないため、再現者がその計算機を利用できない場合、同一の計算を再現できるとは限らない。

AiiDA は永続性の問題に言及していない。

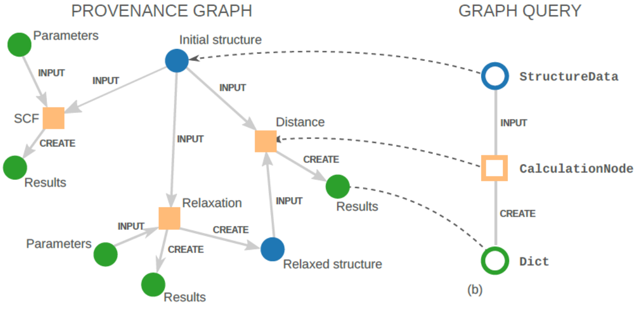

図 4　AiiDA の来歴データモデル[26]

3.6 ReproZip

ReproZip [30]は、コマンドラインプログラムによる実験を完全に再現可能化するスタンドアロンなツールであり、作成者が用いる reprozip と再現者が用いる reprounzip から構成される。作成者が reprozip を介してコマンドを実行すると、reprozip はシステムコールをトレースし、読み込まれたデータファイル、実行可能ファイル（ライブラリ）、環境変数、コマンドラインを取得し、再現に必要なデータをすべて含む「バンドル」を生成する。また、Jupyter Notebook で書かれたプログラムの依存関係を reprozip でバンドル化することも可能である。バンドルを入手した再現者は reprounzip を用いてバンドルを解凍し、実験を再現する。このとき、作成環境と再現環境の違いに応じて、単なるディレクトリ、chroot 環境、コンテナ（Docker）、仮想マシン（Vagrant）の中から適切な解凍形式を再現者が選択する。トレースのデータモデルを図 5 に示す。

ReproZip のようにシステムコールをトレースする方式は、作成者が実行環境を記述する必要がないため、スキルの低い作成者でも確実に再現可能なバンドルを作成でき、依存性の問題を簡単に解決できる。また、実験条件が入力ファイルとコマンドラインで完全に表現されているならば、属人性の問題も解決できる。

ReproZip は永続性の問題を解決しようとするものではない。しかし、外部のシステムに依存しないスタンドアロンなツールであるため、reprounzip が用いる仮想化メカニズム（chroot, Docker, Vagrant）が利用できるかぎり、バンドル化された実験は再現可能であると考えられる。

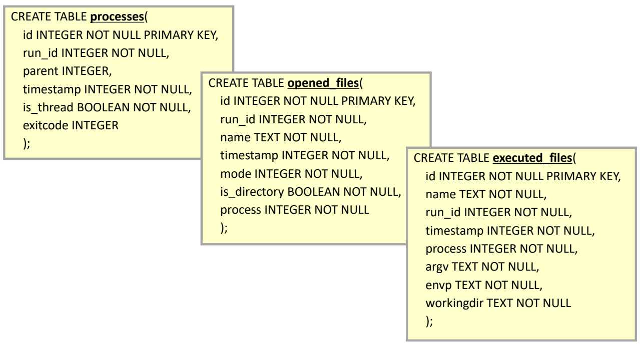

図 5 ReproZip のトレースのデータモデル[31]

3.7 Occam

Occam [32]は、ソフトウェアと長期的な保存と再利用に焦点を当てたオープンソースの PaaS である。ユーザーは、ソースコード、ビルド環境（x86-64, Linux, gcc など）、実行環境（x86-64, Linux, Python、依存パッケージなど）、その他のリソース（必要となるファイル）を指定して「オブジェクト」を作成する。オブジェクトは必要に応じてビルドされ、ビルド時に外部からダウンロードされたパッケージは Occam 内のリポジトリにミラーされる。作成したオブジェクトは Occam 内の VM 上で実行できる。入出力フォーマットが合致する複数のオブジェクトを連結した「ワークフロー」を定義して実行することもできる。他のユーザーは、Occam 上で既存のオブジェクトを検索、再利用できる。オブジェクトがもつデータモデルを図 6 に示す。

Occam は、バイナリやコンテナイメージを保存するのではなく、実行環境を含むソフトウェアのソースコードとビルド手順を保存し、いつでも再構築可能（rebuildable）とすることを重視している点が特徴的である。依存ライブラリ等を外部リポジトリからダウンロードした場合、そのミラーを内部に保存する。作成者と再現者にコンピュータシステムの完全な知識があることを前提として、属人性と依存性の問題をソースコードレベルで検証・修正可能とし、それをもってソフトウェアの永続性を担保しようとする野心的なシステムと言える。

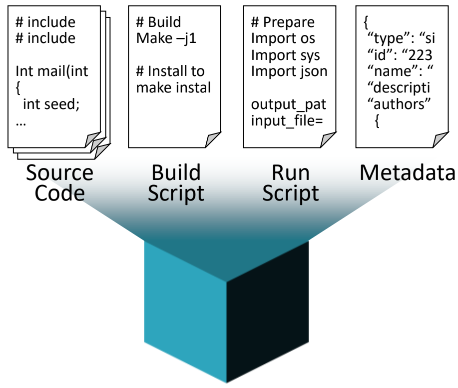

図 6 Occam のオブジェクトのデータモデル[33]

3.8 HubZero

HUBzero [34]は、分野ごとの研究・教育プラットフォームを構築するためのオープンソースのフレームワークである。ナノテクノロジー分野向けの nanoHUB.org [35]、地理空間情報分野向けの MyGeoHub [36]など、44 のサイトが HUBzero を用いて構築・運営されている[37]。HUBzero の中心的な機能は、GUI ベースのインタラクティブな解析ツールを HUBzero のサーバー上で実行し、その画面を VNC でクライアントに送り、ブラウザ上に表示する機能である。計算量の多いボリュームレンダリングを専用のクラスタで実行し、それをブラウザ経由で提供するアーキテクチャは、開発が始まった 2002 年時点において先進的だった。現在では Jupyter Notebook や RStudio も利用できる。

HUBzero にはデータリポジトリとツールリポジトリがあるが、両者を結びつける機能は特にないようである。ユーザーはファイルをデータリポジトリからダウンロードし、サーバー上のホームディレクトリに SFTP や WebDAV でアップロードする。ツールはホームディレクトリ上のファイルを読み書きできる。

HUBzero は、プログラムとその実行環境を OpenVZ コンテナとしてサーバー側で保守管理することで依存性の問題を回避する。HUBzero 本体は属人性の問題を解決せず、実

験の再現性をどのように担保するかは解析ツールの実装次第である。なお、解析ツールはバックエンドに統合された Pegasus ワークフローエンジンの機能を利用できる。

HUBzero は永続性の問題を解決しない。なお、HUBzero の開発チームは 2019 年、OneSciencePlace (OSP) と呼ばれる新たなフレームワークの開発を始めた[38]。OSP では主に資金の観点から永続性の問題に取り組むとしている。

3.9 Software Heritage

Software Heritage は、すべてのオープンなソフトウェアを将来世代に向けて収集、保存、共有するアーカイブである。財団を設立し、UNESCO と協定を結び、マルチステークホルダーな分散ストレージインフラを構築するなど、永続性に対する注力の度合いは他のプロジェクトと一線を画する。

Software Heritage は、インターネット上に公開されているコードリポジトリを定期的にクロールし、コンテント(ファイル内容)、ディレクトリ、リビジョン（コミット)、リリース(タグ)を収集する。同時に、オリジン(取得元 URL)、プロジェクト、スナップショット（ブランチ）を来歴情報として記録する。収集元のリストは GitHub や GitLab などのコードリポジトリと Debian や PyPI などのパッケージリポジトリを含み、順次拡充されている。収集された各オブジェクトは、SHA-1 ハッシュと付加情報からなる Software Heritage ID (SWHID) が永続的識別子として付与され、コンテントを葉とする Merkle DAG（ハッシュ木）構造に編成される。そして、コンテントは KVS に、それ以外の情報は RDB に保存される。保存されたデータは API を用いて検索、取得できる。Software Heritage のデータモデルを図 7 に示す。

Software Heritage は永続性の問題に特化した取り組みであり、研究再現性に関する属人性と依存性の問題を直接解決するものではない。

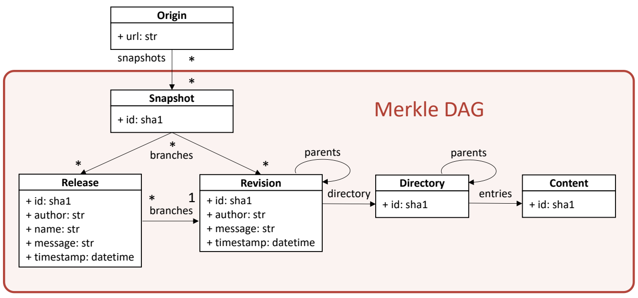

図 7　Software Heritage のデータモデル[39]

4. まとめと考察

本稿では、計算科学の実験を再現可能化する種々のプラットフォームをサーベイした。各プラットフォームが持つデータモデルを図 8 にまとめる。

	要素	概要
(a) Binder	code files	jupyter notebook形式ファイル
	dependency file	環境構築に必要なライブラリの依存関係を記述するrequirements.txt, install.R, runtime.txtなど
(b) Code Ocean	metadata	概要、著者などのメタデータ
	environment	カプセルの計算環境を再構築するためのDockerfile
	code	作成したソースコード
	data	データ
(c) Whole Tale	format	Taleのフォーマットのバージョン
	metadata	タイトルや著者などの9種類からなるメタデータ
	data	データを取得するURLを含むメタデータ
	files	コードや環境変数などのファイルを取得するURLを含むメタデータ
	environment	実行環境を構築するために必要な情報のメタデータ
(d) ReproZip	process	pidと紐づいたデータ解析中の全てのプロセス情報
	opend_files	プロセスによって開かれたファイルの情報
	executed_files	プロセスによるファイルの実行に関する情報
(e) Occam	Source Code	ソースコード
	Build Script	ビルドを実行するためのファイル
	Run Script	再現を実行するためのファイル
	Metadata	タイトル、概要、著者などのメタデータ。ビルドや実行に関する依存関係、インストール、実行方法などの情報も含む
(f) AiiDA	Workflow Node	作成者、ラベル、概要、実行環境、他のノードとのリンク情報などのメタデータ。ワークフロープロセスの実行
	Calculation Node	上記メタデータと、計算プロセスの実行
	Data	上記メタデータと、計算パラメタや結果も含むデータ
(g) KBase	Workspace	意味のあるObject群とその関係の集合
	Object	名前や利用者などのメタデータに加えて、VersionやRelationに関する情報を包含したもの
	Version	あるObjectの同一Workspace内での変更履歴
	Relation	Object間の関係のDependency reference, Provenance reference, Copied fromによる表現
(h) Software Heritage	Origin	URL
	Snapshot	開発プロジェクト全体のスナップショット
	Release	ソフトウェアの各リリース情報
	Revision	ソフトウェアの各コミット情報
	Directory	各Revisionに対応するソースコードのディレクトリ情報とそれに関するメタデータ
	Content	各Revisionに対応するソースコードファイル

図 8　各システムのデータモデルの要約

最後に、2 章で提示した 3 つの論点に沿って各プラットフォームの特長を整理し、分野中立的な研究再現性プラットフォームに適したアプローチを考察する。

4.1 依存性の問題への対処

依存性の問題を解決する方策としては、実行環境の仮想化がほとんどのプラットフォームで採用されている。仮想環境を再現可能とするために、Binder のように環境構成情報（Dockerfile、パッケージリストなど）を作成者が記述する方式と、ReproZip のように実行トレースを自動的に記録して所要のバイナリファイルを保存する方式がある。前者の方式は、再構築時に外部のパッケージリポジトリを参照することが問題となりうる。例えば、将来パッケージリポジトリが廃止されたり、パッケージのアップデートによって挙動が変わったりすると再現性が損なわれる。後者の方式は、外部のリポジトリに依存せず単体で再現性を担保で

Vol.2020-IOT-51 No.11
Vol.2020-SPT-39 No.11
2020/9/4

きる反面、再現された仮想環境には最小限のバイナリしか含まれておらず、再利用性に乏しい。したがって、査読者が実験結果を検証する目的には適しているとしても、再現者が発展的な研究を始めるためにプログラムを改変する目的には適さない。

我々が目指す研究再現性プラットフォームは分野の垣根を超えた発展的な研究の促進を意図していることから、再現された実行環境を容易に再利用できるパッケージリスト方式が適していると考える。パッケージリストのデータモデルとしては、Binder のようにパッケージ管理システムの規格をそのまま利用する設計と、RO-Crate [40]のような標準規格に依拠する設計が考えられる。

4.2 属人性の問題への対処

属人性の問題を解決する方策としては、データ来歴やコマンド履歴を自動的に記録する方式が存在する。ワークフローエンジンの利用も解決策のひとつと言える。AiiDA のようにデータ来歴とワークフローの実行履歴を自動的に記録するプラットフォームを使えばヒューマンエラーを完全に排除でき、属人性の観点からは最も望ましい。サーベイで取り上げた KBase と AiiDA のほかにも、生物学分野で使われる ISA tools [41]や遺伝学分野で使われる GenePattern [42]のように、分野ごとの特性に応じた実験再現性のための来歴データモデルが存在する。来歴データモデルの標準規格として PROV [43]が存在する。

分野中立的な研究再現性プラットフォームを新たに設計するならば、すでに確立したワークフローを持つ大規模な研究分野を対象とするよりも、まだ属人的な作業に依存している中小規模の研究分野を対象とする方が、データ中心科学の方法論を普及させるのに効果的である。手作業に慣れた研究者に最初に普及させる方式としては、データ解析の自動化によってヒューマンエラーを排除する方式よりも、研究者の注意力をある程度信用する文芸的コンピューティング方式が適していると考える。なかでも、Jupyter Notebook 形式がオープンなデータモデルとして最も汎用的で発展性があると考えられる。

4.3 永続性の問題への対処

永続性の問題を解決する方策としては、プログラムと実行環境の実体を（外部リポジトリを参照するのではなく）当該システム内部にアーカイブする方式が有望である。このとき、ReproZip のように実行可能バイナリをアーカイブする方法では、再現された実行環境の再利用性に乏しく、また、ハードウェアへの依存性の問題が残る。新たなハードウェア・アーキテクチャが次々に出現するなかで、作成から再現まで 10 年以上のタイムラグを想定して可搬性と再利用性を担保するには、プログラムと実行環境をソースコードの形でアーカイブする方式が望ましい。ソースコードが残っていれば全文検索が可能であり、アーキテクチャの移行に再ビルドで対処でき、古いプログラムのバグを修正して再利用することも可能である。仮にビルド環境が失われたとしても、少なくとも人間が読んで理解することができる。Software Heritage と Occam がソースコードの保存に注力しているのはこのような理由による。バイナリではなくソースコードを保存する方針は、10 年保存を原則とする研究データよりも長期間にわたってプログラムが実行可能であり続けるために必要であり、また、オープンサイエンスに関するブダペスト宣言[44]の趣旨にも合致する。

ソースコードを保存するためのデータモデルとしては、Software Heritage が開発した Merkle DAG 方式が優れていると考える。NII RDC として永続性の問題に対処しようとするならば、Software Heritage のミラーを保持することが最も効果的な方策と考えられる。ソースコードの保存にとどまらず、より包括的な観点から永続性を担保するには、古いハードウェアを保存する活動[45][46]を研究再現性の一環として支援することも必要であろう。

永続性の問題を複雑にする要因として知的財産権の問題がある。オープンデータとオープンソースソフトウェアだけを用いた研究であれば誰でも問題なく再現できるが、非公開のデータやプロプライエタリなソフトウェアに依存する研究成果を再現するには、原則として、再現者がそれらの利用権を得る必要がある。しかし、作成から再現まで数十年のタイムラグがある場合、データの所有者が消滅したりソフトウェアが入手不能になったりして、研究成果を合法的に再現できなくなることが容易に想像できる。これらは技術的というより法的な問題であり、研究再現性に関する先行研究においても有効な解決策が見いだされていないようである。なお、非公開のデータに関しては、秘密計算技術を用いて元のデータを秘匿化したまま研究成果を再現するアプローチも検討に値する。

4.4 運用時の問題への対処

以上、本稿では汎用的なデータ解析サービスが持つべきデータモデルについて述べたが、運用時にはデータモデルのみならずユーザーへのサポートや運用体制などの問題への対処も必要である。本稿では、属人性の問題への対処として研究者の注意力をある程度信用する方式が適していると考えるが、それには研究者自身が研究プロトコルを確立し、再現可能化のノウハウを蓄積できるような支援が必要となる。例えば、研究者間で分野に特化した Notebook をテンプレートとして蓄積・共有する取り組みや、大学や研究所による研修を実施するための支援機能などを検討する必要があるだろう。

参考文献

[1] M. Baker and D. Penny, “Is there a reproducibility crisis?,” *Nature*, vol. 533, no. 7604, pp. 452–454, 2016, doi: 10.1038/533452A.
[2] R. D. Peng, “Reproducible Research in Computational Science,” *Science (80-.).*, vol. 334, no. 6060, pp. 1226–1227, Dec. 2011, doi: 10.1126/science.1213847.
[3] 国際的動向を踏まえたオープンサイエンスに関する検討会, “我が国におけるオープンサイエンス推進のあり方について～サイエンスの新たな飛躍の時代の幕開け～,” 2015. https://www8.cao.go.jp/cstp/sonota/openscience/.
[4] X. Chen *et al.*, “Open is not enough,” *Nat. Phys.*, vol. 15, no. 2, pp. 113–119, Feb. 2019, doi: 10.1038/s41567-018-0342-2.
[5] V. Stodden *et al.*, “Enhancing reproducibility for computational methods,” *Science (80-.).*, vol. 354, no. 6317, pp. 1240–1241, Dec. 2016, doi: 10.1126/science.aah6168.
[6] K. Chug and R. J. Sethi, “Collaboration in Computer Vision Using Scientific Workflows,” Mar. 2017, pp. 564–567, doi: 10.1109/cts.2016.0104.
[7] “ACM Transactions on Mathematical Software.” https://dl.acm.org/journal/toms.
[8] “PLOS ONE.” https://journals.plos.org/plosone/s/materials-and-software-sharing.
[9] ACM, “SIGMOD 2019 Reproducibility.” http://db-reproducibility.seas.harvard.edu/.
[10] O. Mesnard and L. A. Barba, “Reproducible and replicable CFD: it’s harder than you think,” May 2016, Accessed: Aug. 10, 2020. [Online]. Available: http://arxiv.org/abs/1605.04339.
[11] 日本学術会議, “科学研究における健全性の向上について,” 2015. http://www.scj.go.jp/ja/info/kohyo/pdf/kohyo-23-k150306.pdf.
[12] 瀬口昌久, “研究データの適正な管理の現状と課題,” 技術倫理研究, vol. 14, pp. 1–30, 2017, [Online]. Available: http://id.nii.ac.jp/1476/00006260/.
[13] ACM, “Artifact Review and Badging.” https://www.acm.org/publications/policies/artifact-review-badging.
[14] *Reproducibility and Replicability in Science*. Washington, D.C.: National Academies Press, 2019.
[15] S. L. McArthur, “Repeatability, Reproducibility, and Replicability: Tackling the 3R challenge in biointerface science and engineering,” *Biointerphases*, vol. 14, no. 2, p. 020201, Mar. 2019, doi: 10.1116/1.5093621.
[16] H. E. Plesser, “Reproducibility vs. Replicability: A Brief History of a Confused Terminology,” *Front. Neuroinform.*, vol. 11, Jan. 2018, doi: 10.3389/fninf.2017.00076.
[17] T. Kluyver *et al.*, “Jupyter Notebooks - a publishing format for reproducible computational workflows,” in *ELPUB*, 2016, pp. 87–90.
[18] P. Jupyter *et al.*, “Binder 2.0 - Reproducible, interactive, sharable environments for science at scale,” 2018, pp. 113–120, doi: 10.25080/Majora-4af1f417-011.
[19] A. Clyburne-Sherin, X. Fei, and S. A. Green, “Computational Reproducibility via Containers in Psychology,” *Meta-Psychology*, vol. 3, p. 892, Nov. 2019, doi: 10.15626/mp.2018.892.
[20] A. Brinckman *et al.*, “Computing environments for reproducibility: Capturing the ‘Whole Tale,’” *Futur. Gener. Comput. Syst.*, vol. 94, pp. 854–867, 2019, doi: 10.1016/j.future.2017.12.029.
[21] The Whole Tale, “Tale Serialization Format.” https://wholetale.readthedocs.io/en/stable/development/mockups/tale-serialization/.
[22] A. P. Arkin *et al.*, “KBase: The United States department of energy systems biology knowledgebase,” *Nat. Biotechnol.*, vol. 36, no. 7, pp. 566–569, 2018, doi: 10.1038/nbt.4163.
[23] KBase, “Data Type Catalog.” https://narrative.kbase.us/#catalog/datatypes.
[24] KBase, “Workspace fundamentals.” https://kbase.us/services/ws/docs/fundamentals.html.
[25] “KBase SDK Documentation.” https://kbase.github.io/kb_sdk_docs/overview.html.
[26] S. P. Huber *et al.*, “AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance,” 2020, [Online]. Available: http://arxiv.org/abs/2003.12476.
[27] “AiiDA lab.” https://aiidalab.materialscloud.org/.
[28] L. Talirz *et al.*, “Materials Cloud, a platform for open computational science,” Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.12510.
[29] “Materials Cloud Archive.” https://archive.materialscloud.org/.
[30] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, “ReproZip,” in *Proceedings of the 2016 International Conference on Management of Data - SIGMOD ’16*, 2016, pp. 2085–2088, doi: 10.1145/2882903.2899401.
[31] ReproZip, “Trace Database Schema.” https://docs.reprozip.org/en/1.0.x/traceschema.html.
[32] L. Oliveira *et al.*, “Long-term Preservation of Repeatable Builds in Occam,” *Proc. CANOPIE-HPC 2019 1st Int. Work. Contain. New Orch. Paradig. Isol. Environ. HPC - Held conjunction with SC 2019 Int. Conf. High Perform. Comput. Networking, Storage*, vol. 18, pp. 21–30, 2018, doi: 10.1145/3214239.3214244.
[33] L. Oliveira, D. Wilkinson, D. Mossé, and B. Childers, “Supporting Long-term Reproducible Software Execution,” in *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems - P-RECS’18*, 2018, vol. 18, pp. 1–6, doi: 10.1145/3214239.3214245.
[34] M. McLennan and R. Kennell, “HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering,” *Comput. Sci. Eng.*, vol. 12, no. 2, pp. 48–53, Mar. 2010, doi: 10.1109/MCSE.2010.41.
[35] “nanoHUB,” [Online]. Available: https://nanohub.org/.
[36] “MyGeoHUB.” https://mygeohub.org/.
[37] “HUBzero powered sites.” https://hubzero.org/sites/.
[38] D. Benham and S. Gesing, “HUBzero© Goes OneSciencePlace: The Next Community-Driven Steps for Providing Software-as-a-Service,” in *2019 15th International Conference on eScience (eScience)*, Sep. 2019, pp. 642–643, doi: 10.1109/eScience.2019.00097.
[39] R. Di Cosmo, M. Gruenpeter, and S. Zacchiroli, “Referencing source code artifacts: A separate concern in software citation,” *Comput. Sci. Eng.*, vol. 22, no. 2, pp. 33–43, 2020, doi: 10.1109/MCSE.2019.2963148.
[40] ResearchObject, “Research Object Crate (RO-Crate).” https://w3id.org/ro/crate.
[41] “ISA Model & Serialization Specifications.” https://isa-tools.org/format/specification.html.
[42] Y. Huang and R. Gottardo, “Comparability and reproducibility of biomedical data,” *Brief. Bioinform.*, vol. 14, no. 4, pp. 391–401, Jul. 2013, doi: 10.1093/bib/bbs078.
[43] W3C, “PROV.” https://www.w3.org/TR/prov-overview/.
[44] “ブダペスト・オープンアクセス・イニシアティヴから 10 年：デフォルト値を「オープン」に,” 2012. https://www.budapestopenaccessinitiative.org/boai-10-translations/japanese-translation-1.
[45] “Living Computers Museum + Labs.” https://www.livingcomputers.org/.
[46] “情報処理技術遺産.” http://museum.ipsj.or.jp/heritage/.