Preprint Cambridge Centre for Computational Chemical Engineering ISSN 1473 – 4273

# A Derived Information Framework for a Dynamic Knowledge Graph and its Application to Smart Cities

Jiaru Bai [1], Kok Foong Lee [2], Markus Hofmeister [1,3], Sebastian Mosbach [1,3], Jethro Akroyd [1,3], Markus Kraft [1,3,4,5]

released: February 6, 2023

[1] Department of Chemical Engineering and Biotechnology
University of Cambridge
Philippa Fawcett Drive
Cambridge, CB3 0AS
United Kingdom

[2] CMCL Innovations
Sheraton House
Cambridge
CB3 0AX
United Kingdom

[3] CARES
Cambridge Centre for Advanced Research and Education in Singapore
1 Create Way
CREATE Tower, #05-05
Singapore, 138602

[4] School of Chemical and Biomedical Engineering
Nanyang Technological University
62 Nanyang Drive
Singapore, 637459

[5] The Alan Turing Institute
London
NW1 2DB
United Kingdom

Preprint No. 302

UNIVERSITY OF CAMBRIDGE

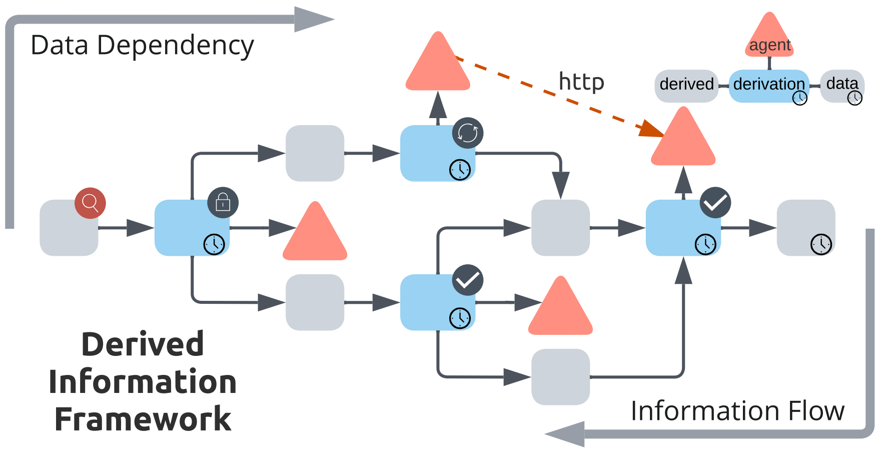
*Keywords:* Knowledge graph, derived information, provenance, directed acyclic graph, smart cities

**Edited by**
Computational Modelling Group
Department of Chemical Engineering and Biotechnology
University of Cambridge
Philippa Fawcett Drive
Cambridge, CB3 0AS
United Kingdom

**E-Mail:** mk306@cam.ac.uk
**World Wide Web:** `https://como.ceb.cam.ac.uk/`

## Abstract

In this work, we develop a derived information framework to semantically annotate how a piece of information can be obtained from others in a dynamic knowledge graph. We encode this using the notion of a "derivation" and capture its metadata with a lightweight ontology. We provide an agent template designed to monitor derivations and to standardise agents performing this and related operations. We implement both synchronous and asynchronous communication modes for agents interacting with the knowledge graph. When occurring in conjunction, directed acyclic graphs of derivations can arise, with changing data propagating through the knowledge graph by means of agents' actions. While the framework itself is domain-agnostic, we apply it in the context of Smart Cities as part of the World Avatar project and demonstrate that it is capable of handling sequential events across different timescales. Starting from source information, the framework automatically populates derived data and ensures they remain up to date upon access for a potential flood impact assessment use case.



## Highlights

- An ontology to represent derived information in a knowledge graph is developed.
- Generic agent templates for handling the derivation process are provided.
- The fully automated framework populates directed acyclic graphs of derived information.
- Information updates propagate through the knowledge graph upon access.
- The framework is applied to assess the impact of a potential flooding event.

# Contents

## 1 Introduction

Inspired by Semantic Web technology, knowledge graphs are gaining popularity both in enterprise applications [43] and research fields [30]. They are seen as a suitable approach to integrating diverse information sources and fostering common understanding among domain experts [27, 34]. Some renowned examples of static knowledge graphs are DBpedia [37] and Wikidata [45].

The dynamic aspect of knowledge graphs has recently been studied [1, 6], where they are used as hubs for integrating complex systems of software agents, connecting different domains in specific use cases. This allows for what-if scenario analysis and automated decision-making that mimics human behaviour. This is a step towards the original vision of the Semantic Web, which is a fully annotated web of machine-readable data that can be processed autonomously by software agents [7, 29]. However, this also highlights the need for robust methods to manage the changes and interdependencies in the complex information network, especially in a data-rich world which is rife with misinformation.

A key enabling factor identified by the community is provenance [51], *i.e.* where a piece of information originates from and how it came about. Provenance covers many aspects, including published literature, the wider internet, or data directly acquired through a measurement device. A number of provenance ontologies have been developed in the literature, for example, PAV [10], W3C Provenance Ontology (PROV-O) [36], Dublin Core Terms (DC Terms) [15], Bibliographic Ontology (BIBO) [14], and Open Provenance Model Ontology (OPMO) [41]. For a comprehensive review of developments in provenance data models, interested readers are referred to Sikos and Philp [47].

In dynamic knowledge graphs, one can consider a specific sub-problem of provenance, namely when some pieces of information are directly calculated from, or derived from, other pieces of information by software agents, all of which are already stored in the knowledge graph. When multiple pieces of information depend on each other in a certain way, the resulting cascade of information is progressed in time via a series of coordinated agent communications, where the calculated values of one process are inputs for other subsequent processes. To make the knowledge graph truly dynamic, the system should also enable automated propagation of perturbations in source input data. This introduces another point of view – caching. In addition to providing functions (agents) to calculate a result, that result is stored, *i.e.* cached, in the knowledge graph. Many of the technical challenges are similar, such as being able to detect when the cache is out of date, and providing a function to update, refresh, or recalculate the cache.

Any approach to solving this problem can benefit from lessons learned in other fields. In scientific computing, such a composition of computing tasks is referred to as *workflow* or *pipeline* [17, 50]. Workflows are typically expressed as directed acyclic graphs (DAGs), where the nodes represent tasks and edges represent data flows [38]. Each task can only be started if all its precedent tasks are completed. There are many different workflow management systems (WMSs) that can be used to orchestrate these tasks, ranging from traditional paradigms like Pegasus [16, 18] and Kepler [4] to modern approaches like Apache Airflow [48], Nextflow [22], and Lightning AI [24], to name a few. Some studies have focused on adaptive workflows, where changes in the input data may be incorporated into computation on-the-fly to provide real-time responses to dynamic events [40]. However, these systems often rely on heterogeneous and unstructured data models without semantic annotations [19], which can make it difficult to achieve interoperability across different systems, impeding the unification of the workflow community [12].

Another area which we may learn from is microservice architecture [23]. It is a service-oriented architecture (SOA) that emphasises loose coupling and high cohesion. It involves having each service in the ecosystem be independently developed, deployed, and maintained by a small dedicated team. When an *event* occurs, which is a similar concept as an execution instance of one pre-defined workflow, microservices are composed and coordinated via message-passing. This architecture places no restrictions on developers regarding the technology used to implement each microservice as long as a unified interface is agreed upon. Nonetheless, it places significant demands on the network to ensure successful communication. In this paradigm, Distributed Application Runtime (Dapr) [13] uses a sidecar to simplify communication via direct or event-based publish/subscribe messaging, unifying both modes of communication on the same platform.

Taking notes from these advents, we may summarise and suggest a knowledge-graph-native solution. By design, data in the knowledge graph can be uniquely identified via Internationalised Resource Identifiers (IRIs). All the active agents share the same world-view once granted access privileges. Analogous to the *message bus* in the microservice architecture, the communication between agents operating on the knowledge graph can be delegated via serving the correct IRIs. This eliminates the necessity for large peer-to-peer data transfer. It can be further combined with the idea learned in the scientific workflow to model computation dependencies as DAGs in the knowledge graph. By encoding agents' messages as provenance records, we remove the need for direct agent-to-agent communication and allow information to travel through the knowledge graph. This decouples the system and allows for a distributed ecosystem of agents without the need for them to be aware of each other's existence.

As a "digital twin" of the world that is realised as a dynamic knowledge graph [1], the World Avatar is thought to be an appropriate candidate for evaluating the implementation of this technology. As it stands, there is an interwoven network of concepts spanning temporal and spatial scales, extending from molecule [26] and chemical mechanism [25] to laboratory [6], cities [9], and even the national level [1]. The World Avatar is constantly evolving as it is maintained by a network of active agents who regularly input new data and update existing data. To effectively manage the actions of these agents and ensure accurate tracking of their activities, an overarching architecture is required.

The **purpose of this paper** is to demonstrate a proof-of-concept for a technology-agnostic implementation of a derived information framework for dynamic knowledge graphs. This derivation framework is a realisation of such a knowledge-graph-native architecture. It uses a lightweight ontology to mark up provenance, an agent template to standardise agent operations, and an automated framework to propagate information changes in a dynamic knowledge graph. The design aims to lower the entry barrier for researchers to model any real-world cascading events with minimum effort by providing a user-friendly template. In particular, we demonstrate this through a flood impact assessment use case within the World Avatar project.

The presentation of this paper is structured as follows. Section 2 situates the work by summarising the lessons we learned through the development of a dynamic-knowledge-graph-based digital twin of the world; section 3 provides the technical details on the complete architecture; section 4 exemplifies the versatility of the framework via a use case in the context of smart cities; and section 5 concludes the work.

## 2 The World Avatar

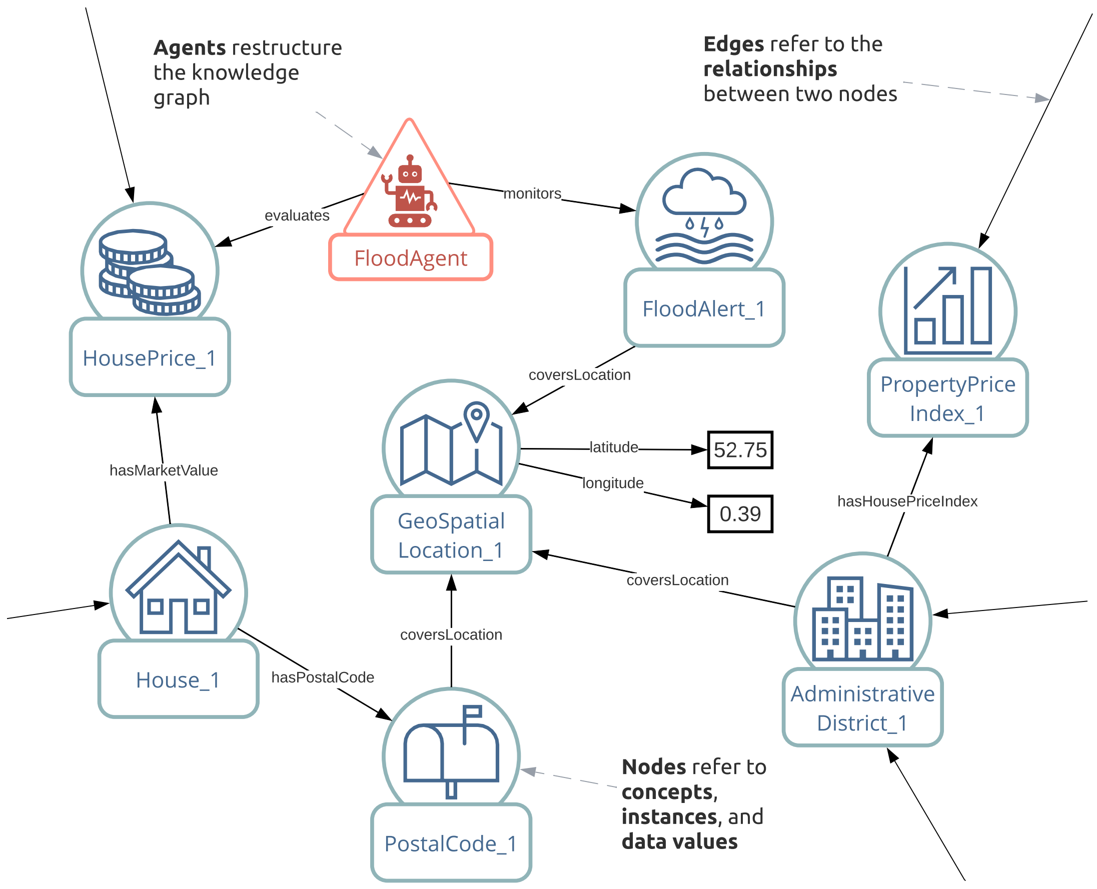


**Figure 1:** *Schematic of the World Avatar knowledge graph. Note that the figure is only illustrative and does not reflect actual data.*

In the World Avatar, the physical world we live in is captured and represented as the *base world*. The hypothetical versions of the world in which certain variables or assumptions are different are represented as *parallel worlds*. These alternative universes are managed by software agents that can perform a variety of tasks. Importantly, the World Avatar views these agents to be part of the knowledge graph, as depicted in Fig. 1. Using this technology stack, the World Avatar is versatile in three aspects: (1) answering cross-domain questions about the base world [3], (2) controlling real-world entities [6, 32], and (3) supporting what-if scenario analysis with parallel worlds [2, 46].

As the World Avatar project continues to evolve, it strives to more accurately represent complex phenomena across spatio-temporal scales. This requires a tool for efficiently coordinating the actions of agents that update and restructure the knowledge graph. Currently, software agents are represented similar to semantic web services [52], which are invoked through HTTP requests. For time-consuming computations, an asynchronous job watcher is available to delegate jobs to high-performance computing (HPC) facilities. This was applied to assess the impact of quantum calculations on the air pollution dispersion [42] and to automate the calibration of combustion mechanisms [5]. These approaches largely adhere to the static remote procedure call paradigm. To fully unlock the potential of the dynamic world model, a more flexible and adaptive architecture that can facilitate autonomous interaction between agents and the knowledge graph is preferred.

The derived information framework described in this work offers such a solution as a first step towards revolutionising the agents' operations in dynamic knowledge graphs.

## 3 Methodology

This section provides an overview of the technical aspects of the derived information framework. We begin by introducing the ontology created for annotating the provenance markup, then presenting the agent template that developers can use as a starting point when developing new agents. Lastly, we discuss a client library, which can be used to manage the derivation instances.

### 3.1 Derivation ontology

We refer to "derivation" as the record for a singular occurrence of the fact that some pieces of information are derived or calculated from some other pieces of information. The term "derivation subgraph" is used to describe the subgraph of all derivation-related markup when a collection of interdependent processes is represented. Since the information in a knowledge graph is captured in the format of Subject–Predicate–Object statements (triples), storing the markup to capture this fact can be considered as a way of attaching arbitrary metadata to triples [39]. For that, generic solutions have been developed or are currently in development, such as reification [28] and W3C's RDF-star [49], which at the time of writing is still at the draft stage. While some implementations exist, *e.g.* Blazegraph's Reification Done Right (RDR) [8, 28] or more recently GraphDB [44], at present these are not sufficiently widely supported for a technology-agnostic way of implementing the derivation framework, without tying ourselves to a particular product. Therefore, we choose to state the required metadata explicitly as triples and introduce OntoDerivation as a lightweight ontology to serve this purpose. The terminological component (TBox) is explained first, followed by an example instantiation of the assertional component (ABox). The connection between OntoDerivation and OntoAgent [52], an ontology used to define the capabilities of agents, is further demonstrated by how they may be used in conjunction to govern agent actions.

### 3.1.1 OntoDerivation TBox

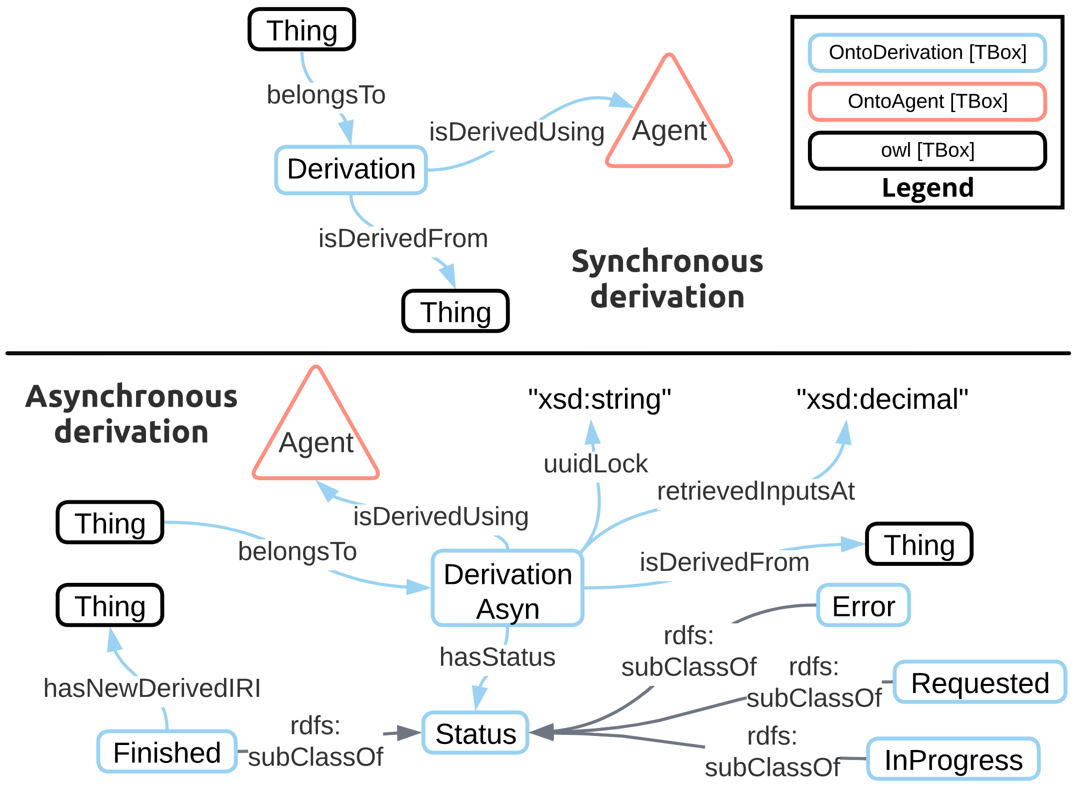


**Figure 2:** *Concepts and relationships of the OntoDerivation ontology. All classes and properties belong to the OntoDerivation namespace unless stated otherwise (for namespace definitions see A.1).*

Figure 2 depicts two types of derivation, namely `Derivation` and `DerivationAsyn`. They are classified as synchronous (`Derivation`) and asynchronous (`DerivationAsyn`) to accommodate situations that respond in different timescales when a request is received. The synchronous mode communicates via the endpoint exposed by the software agent. It is thus faster and hence intended for applications demanding real-time responses. The asynchronous mode communicates exclusively through the knowledge graph. It has the advantage of recording each stage of the operation in the knowledge graph, but it is slower and hence better suited to relatively expensive jobs.

The information dependencies of a derivation are consistently marked regardless of the communication protocol, with the derived information (outputs) `belongsTo` the derivation, which itself `isDerivedFrom` some source information (inputs) and `isDerivedUsing` an agent defined in OntoAgent [52]. It is worth noting that there is no limit on the number of inputs or outputs for a derivation, but one output entity cannot `belongsTo` more than one derivation instance. Both source and derived information are abstracted using `owl:Thing`, so it is also possible for an input of a derivation to be part of another derivation instance, meaning that the input is a piece of derived information itself and `belongsTo` another derivation.

To support asynchronous operation, the concept `Status` is introduced to mark the state of an asynchronous derivation with the available options of `Requested`, `InProgress`, `Finished` and `Error`. The data property `retrievedInputsAt` records the timestamp when the inputs for the derivation were read in order to start the computation, which will later be used to update the timestamp for the derivation instance. The data property `uuidLock` uniquely identifies the agent thread that is processing the derivation and prevents any amendment from other threads that do not hold the correct key. This ensures thread-safe operations when multiple threads are employed to boost the throughput of derivation processing. A specific object property called `hasNewDerivedIRI` is used at the `Finished` status to temporarily link any newly derived information. These output entities will eventually be reconnected to the derivation instance after the agents clean up the status. Like the final outputs, there is no limit on the number of new entities that can be connected through `hasNewDerivedIRI` for each derivation instance.

A description logic representation of OntoDerivation ontology is provided in A.2.

### 3.1.2 OntoDerivation ABox

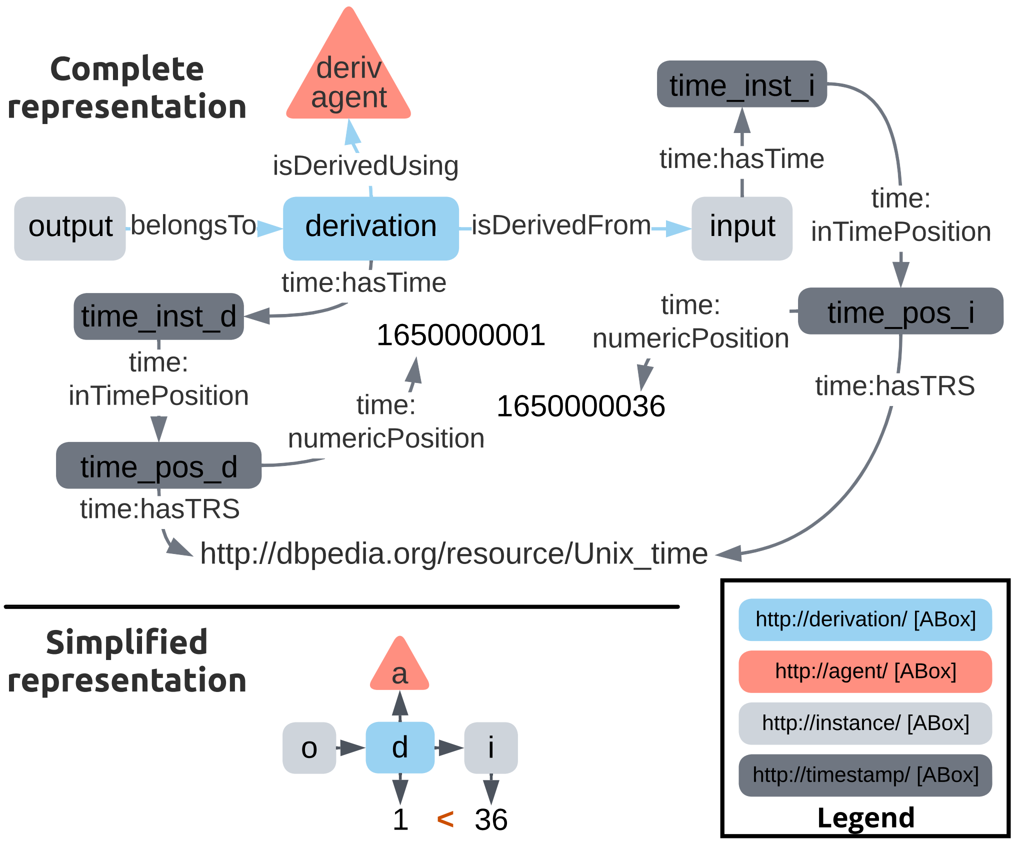


**Figure 3:** *An example derivation instance fully annotated with metadata and its simplified representation, which will be used throughout the rest of this paper. All properties belong to the OntoDerivation namespace unless stated otherwise (for namespace definitions see A.1).*

Figure 3 exemplifies an instantiated derivation instance and its simplified representation. Upon initialisation, each derivation is annotated with a timestamp following the W3C standard [11]. In the rest of this paper, simple integers will be used instead of the actual (Unix) timestamp for readability. Over the lifecycle of a derivation, this timestamp is used to assess whether it is out-of-date. In this example, as the timestamp of derivation is smaller than that of its source information, *i.e.* $1 < 36$, we conclude that this derivation is outdated and that, hence, an update of the derived information is required. It should be noted, however, that the timeliness of the derived information is reflected by the timestamp of the derivation instance it `belongsTo`, and hence does not contain a timestamp itself. As a result, additional criteria for determining whether a derivation is out-of-date should be used when any of its inputs is part of another derivation instance, meaning the existence of a derivation subgraph comprised of several connected derivations. More information on its implementation is provided in section 3.3.

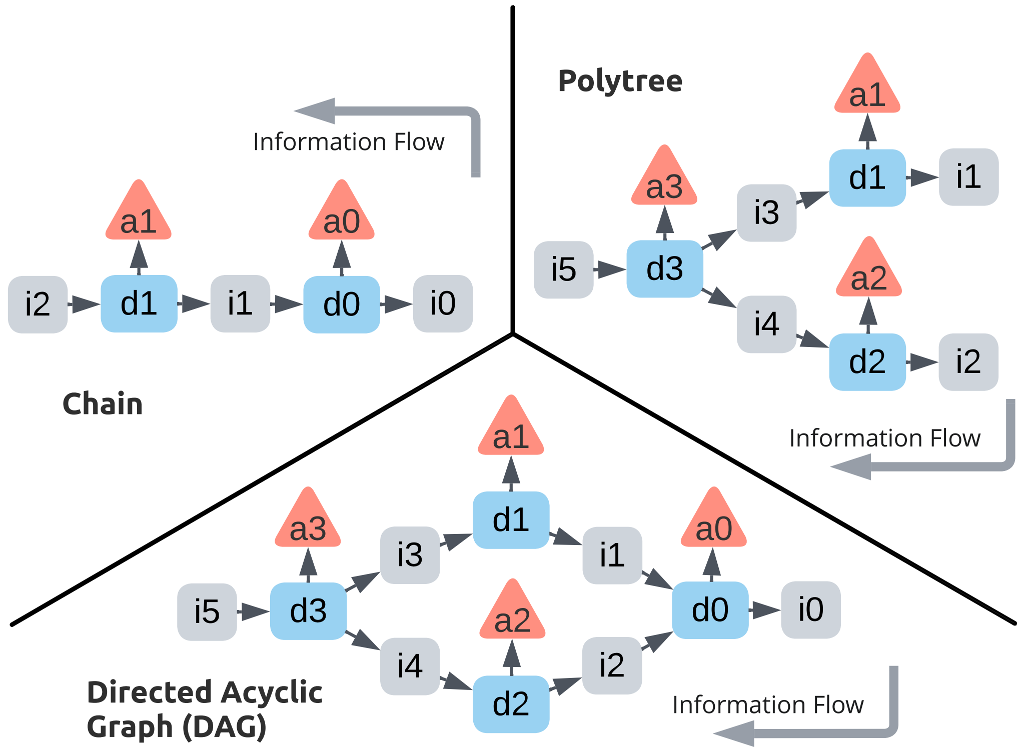


**Figure 4:** *Derivation subgraph structures as directed acyclic graphs (DAGs) of varying generality. The arrows between instances indicate the markup for data dependencies. The "information" flows in the opposite direction, i.e. from right to left.*

Figure 4 illustrates three levels of generality in a derivation subgraph, from basic linear *chain* to non-linear *polytree* to generic *directed acyclic graph*. Unlike in scientific workflows, where the arrows often point in the same direction as the data flow, the markup in the knowledge graph denotes the data dependencies and points to the source of the information. The changes in the source information travel in the opposite direction within the knowledge graph, as illustrated by "information flow". In this context, we define 'upstream' and 'downstream' to refer to the relative location of a derivation instance within that flow. A key feature of the design is that only relevant downstream information is updated when accessed, which will be discussed in more detail in section 3.3.

We emphasise that the primary intent of the derivation framework is to represent logical dependencies as such, as a historical record, rather than consider them as 'steps' in a workflow or algorithm. This implies that, in contrast to workflows [35], cyclic dependencies are not permitted in the derivation framework, as they would amount to logical contradictions. Nonetheless, the framework can of course be used to record the dependencies of pieces of information that were obtained from algorithms containing loops and other circular constructs.

OntoDerivation ontology is designed in a way that is easy to use and easy to query. Tools are provided to automatically generate derivation markup for all three types, and validate the generated derivation subgraph. An example SPARQL query to retrieve all derivations in a given knowledge graph is provided in Query 1.

### 3.1.3 Connection with OntoAgent

In the World Avatar, OntoAgent [52] is used to mark up the input/output (I/O) signature of agents to facilitate agent discovery. This markup points to concepts in domain ontologies and indicates the agent's capabilities. By contrast, OntoDerivation focuses on the instance level, *i.e.* actual data digested and produced by the agents corresponding to each occurrence of computation, revealing the opportunity for employing both ontologies to regulate agent operations.

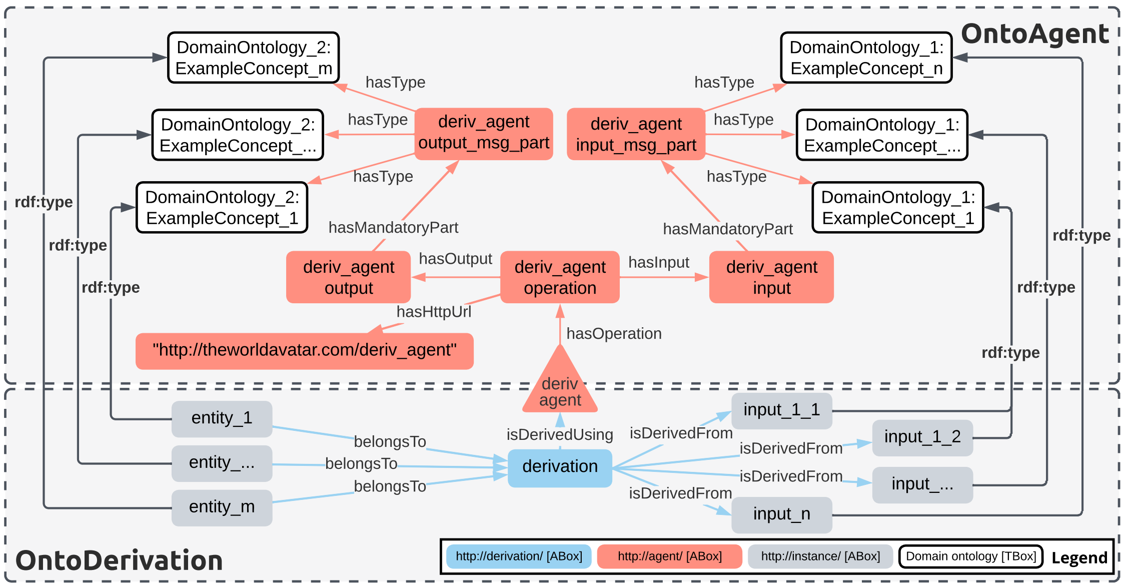


**Figure 5:** *The instantiation that connects OntoDerivation and OntoAgent. The linkage between derivation instances and agent instances is used to regulate agent operations. All object properties belong to the OntoDerivation namespace unless stated otherwise (for namespace definitions see A.1).*

Figure 5 provides an example of instantiation using both OntoDerivation and OntoAgent, where inputs and derived outputs of the derivation instance are both instantiated from the concepts pointed to by the OntoAgent instance. For a given derivation instance, the inputs can be classified into key-value pairs, with the IRI of each concept as the key and the list of instances as the value. For example, the value of the pair with the key *DomainOntology_1:ExampleConcept_1* will be *input_1_1* and *input_1_2*. This design connects the conceptual capability of the agents with the concrete tasks that they are assigned to execute. Following this practice, the development of agents can be focused on the concept level, simplifying the implementation.

## 3.2 Derivation agent

The use of ontological markup to record each step in the process of updating derived information largely restricts the communication an agent needs to perform to the knowledge graph itself, rather than with other agents. We provide an agent template to support this in both synchronous and asynchronous modes. The template makes use of data container classes to host agents' inputs and outputs. These classes are key-value pairs that may be mapped using query 2. Developers are supplied with utility functions to access/validate the mapped inputs and to construct outputs. The agent logic that computes outputs from the inputs is the only code required from the developer. We made the template available in both Java and Python to increase accessibility and depict its unified modeling language (UML) activity diagram in Fig. 6. The essential design elements are elaborated on below.

### 3.2.1 Synchronous communication mode

The synchronous communication mode is realised through direct agent requests/responses. Upon receiving a request for a normal `Derivation`, the agent serialises the request content to an instance of the container class. The time instant is recorded immediately before passing the inputs to be processed by the developers' code. This instant is considered as the timestamp when inputs were read. Once the outputs are constructed, an update operation will be formulated and executed by the agent to update the knowledge graph. If there is no error, the derivation update is considered successful and a response will be returned that includes the produced derived information and the recorded timestamp.

### 3.2.2 Asynchronous communication mode

In the asynchronous communication mode, agents monitor the status of derivations in the knowledge graph and perform any requested tasks. When an agent detects a derivation with the status `Requested`, it first checks if all the data dependencies for that derivation are satisfied using Query 3. If the requirements are met, the agent retrieves the inputs from the knowledge graph, records the current timestamp, and changes the status of the derivation to `InProgress`. The inputs are then passed to the method provided by the developer and transformed into outputs. At this point, the status of the derivation will be changed to `Finished` and the newly derived outputs are temporarily connected to it.

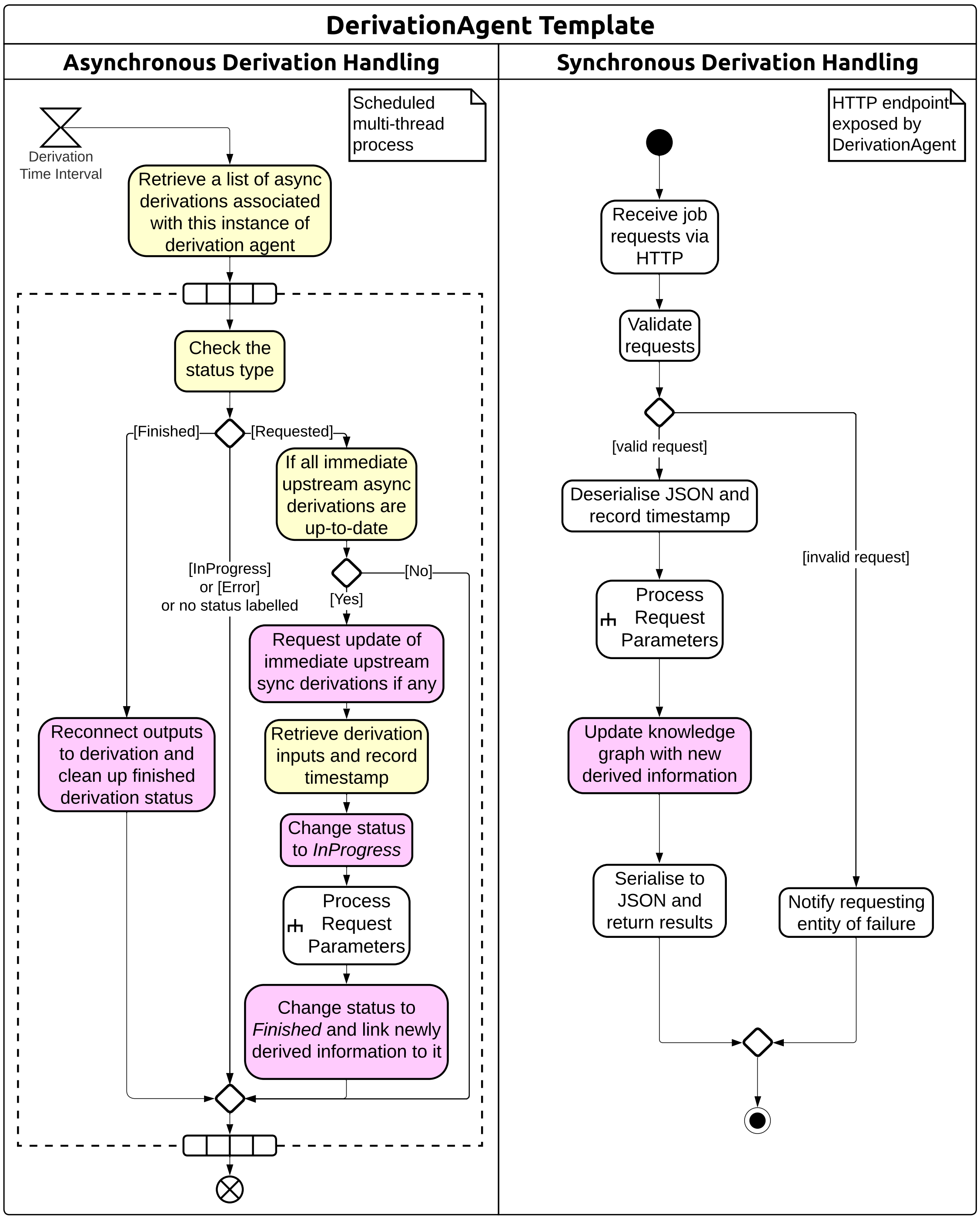

DerivationAgent Template
Asynchronous Derivation Handling
Synchronous Derivation Handling
Scheduled multi-thread process
Derivation Time Interval
Retrieve a list of async derivations associated with this instance of derivation agent
Check the status type
[Finished]
[Requested]
If all immediate upstream async derivations are up-to-date
[InProgress] or [Error] or no status labelled
[No]
[Yes]
Request update of immediate upstream sync derivations if any
Reconnect outputs to derivation and clean up finished derivation status
Retrieve derivation inputs and record timestamp
Change status to InProgress
Process Request Parameters
Change status to Finished and link newly derived information to it
HTTP endpoint exposed by DerivationAgent
Receive job requests via HTTP
Validate requests
[valid request]
Deserialise JSON and record timestamp
[invalid request]
Process Request Parameters
Update knowledge graph with new derived information
Serialise to JSON and return results
Notify requesting entity of failure

**Figure 6:** *UML activity diagram of the derivation agent template supporting both synchronous and asynchronous derivations. Developers need only supply the activity node* `ProcessRequestParameters` *for specific agent capabilities. The yellow- and magenta-shaded actions represent knowledge graph data retrieval and population operations respectively.*

This status refers to a distinction made between a task that has been completed but still requires post-processing or cleaning-up, and a task that is complete in the sense that it requires no further action. It is used to prevent multiple agents from trying to perform the same cleaning-up tasks simultaneously and is removed when the derivation subgraph is tidied up during the next scheduled monitoring period. The cleaning-up process includes deleting the old instances, connecting the new instances with the original derivation and any downstream derivations that exist, removing all the status information, and finally updating the timestamp of the derivation to keep the derivation subgraph current. The monitoring is performed at a scheduled time interval and its frequency can be user-defined. If an error occurs during any operation, the agent changes the status of the derivation to `Error` and records the exception trace.

### 3.2.3 Concurrency and multi-threading

Being a decentralised system by design, the World Avatar contains many agents that are operating on the knowledge graph simultaneously. In situations where multiple agents request updates to the same derivation, the corresponding agent must handle concurrent requests correctly and efficiently. For example, such a situation arises when instance *i1* and *i2* illustrated in the generic form of DAG in Fig. 4 are accessed at the same time. In synchronous communication mode, the current implementation ensures that no duplicated information is added to the knowledge graph through the use of the SPARQL update detailed in Query 4. In asynchronous communication mode, the agent uses the data property `uuidLock` to identify and lock the thread currently handling a derivation, avoiding duplicated execution. These measures ensure that concurrency is handled correctly without sacrificing the high throughput of multithreading, paving the way for distributed agent deployment.

## 3.3 Derivation client

Having established the ontology to capture the derivation process and the agent template to perform the derivation update, we introduce here the derivation client capable of managing the derivation subgraphs. This involves determining if a derivation is out-of-date and, if so, requesting an update. This section discusses three cases in which updates to the derivation subgraph are handled using different communication modes, namely: purely synchronous, purely asynchronous, and mixed-type. For each case, we first present the general algorithm and then provide examples to describe the intended outcome.

### 3.3.1 Purely synchronous update

Determining the timeliness of each derivation and performing the necessary updates in a derivation subgraph is a recursive process. Given any derivation instance where the accessed information is derived from, the framework treats it as root, traverses upstream all the way to the derivation that is derived from source information, *i.e.*, all inputs of whom are not derived from anything, and finally updates derivations backwards. This may be described as a depth-first search (DFS) algorithm, presented in Algorithm 1.

```
Algorithm 1: updateSyncDerivation(rootDerivationIRI)
Input: IRI of the root derivation instance
Result: The root derivation and all its upstream derivations are updated if deemed
        outdated
Create an empty directed acyclic graph $G$;
Cache root derivation $d$ and all its upstream derivations recursively in $G$;
updateSyncDerivation($d$, $G$);

Function updateSyncDerivation($d$, $G$):
    $U \leftarrow d.upstreams()$ ;      /* get immediate upstream derivations */
    if $vertex\ d \notin G.vertices$ then
        Add $d$ as vertex in $G$;
    end
    for $u \in U$ do
        if $vertex\ u \notin G.vertices$ then
            $visited_u \leftarrow false$;
            Add $u$ as vertex in $G$;
        else
            $visited_u \leftarrow true$;
        end
        if $edge\ (d,u) \notin G.edges$ then
            $traversed_{d,u} \leftarrow false$;
            Add $(d,u)$ as edge in $G$ ;      /* will throw error if circular
              dependency detected */
        else
            $traversed_{d,u} \leftarrow true$;
        end
        if $visited_u == false$ and $traversed_{d,u} == false$ then
            updateSyncDerivation($u$, $G$);
        end
    end
    Fire request to update $d$ if it is deemed outdated;
    Update outputs of $d$ in cache;
```

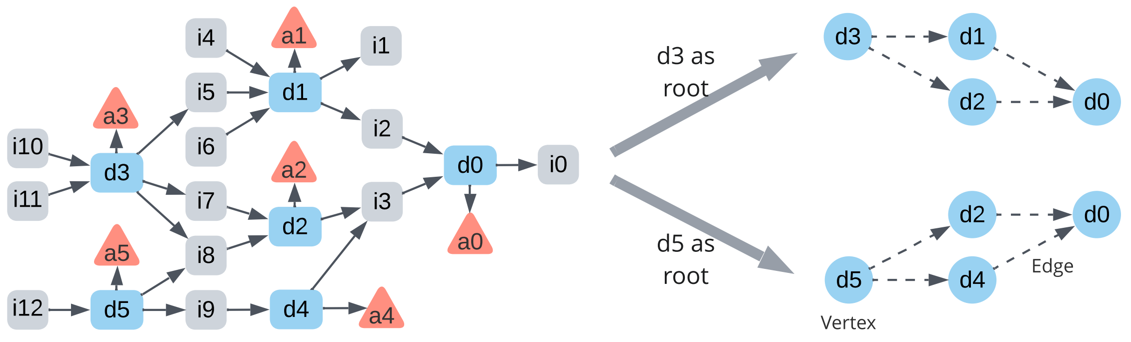


**Figure 7:** *DAGs are used in memory to assist the backtracking of the DFS algorithm when updating the derivation subgraph. The derivation instances are treated as vertices and their connections as edges. Depending on the root derivation chosen, different DAGs can be generated.*

Figure 7 illustrates a notable detail of Algorithm 1 that uses the DAG $G$ to track the traversing of the graph. To do this, the algorithm adds derivation instances as vertices and connections between them as directed edges to $G$. Depending on the root derivation, the DFS algorithm can result in two versions of $G$, leaving out irrelevant parallel branches. The update is carried out only for the derivations in the resulting graph during the DFS algorithm's backtracking. It should also be noted that, the computation only proceeds when both node and edge are previously unseen. For example, regardless of which branch is traversed first (derivation *d1* or *d2*) in the upper resulting graph, derivation *d0* is only visited the first time when branching. This design ensures the relevant upstream information are only visited once to avoid duplication of work.

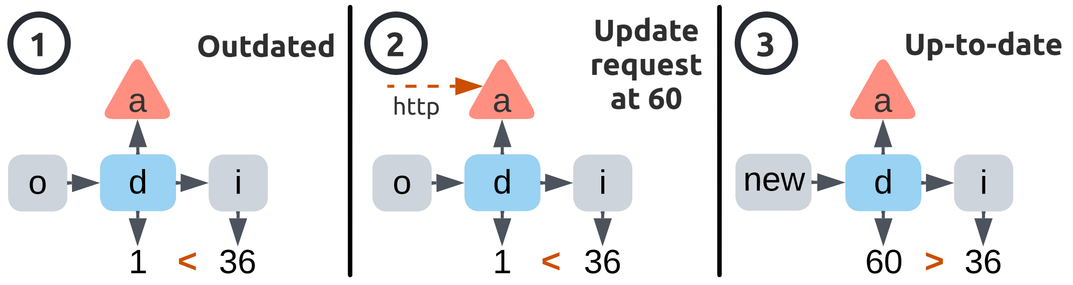


**Figure 8:** *The process of updating a single synchronous derivation.*

Figure 8 illustrates the simplest form of derivation update, *i.e.*, updating one synchronous derivation. For demonstration, we take the simplified representation of derivation expressed in Fig. 3 at timestamp 36 as a starting point. As aforementioned, this derivation is deemed outdated when comparing its timestamp with that of its inputs. Assuming the output information is accessed at 60, the framework fires an update request to the agent associated with the derivation instance. Upon receiving the update request, the agent starts a calculation immediately and updates the output entity in the knowledge graph with the newly derived information. The derivation instance is thus up-to-date and the results are returned.

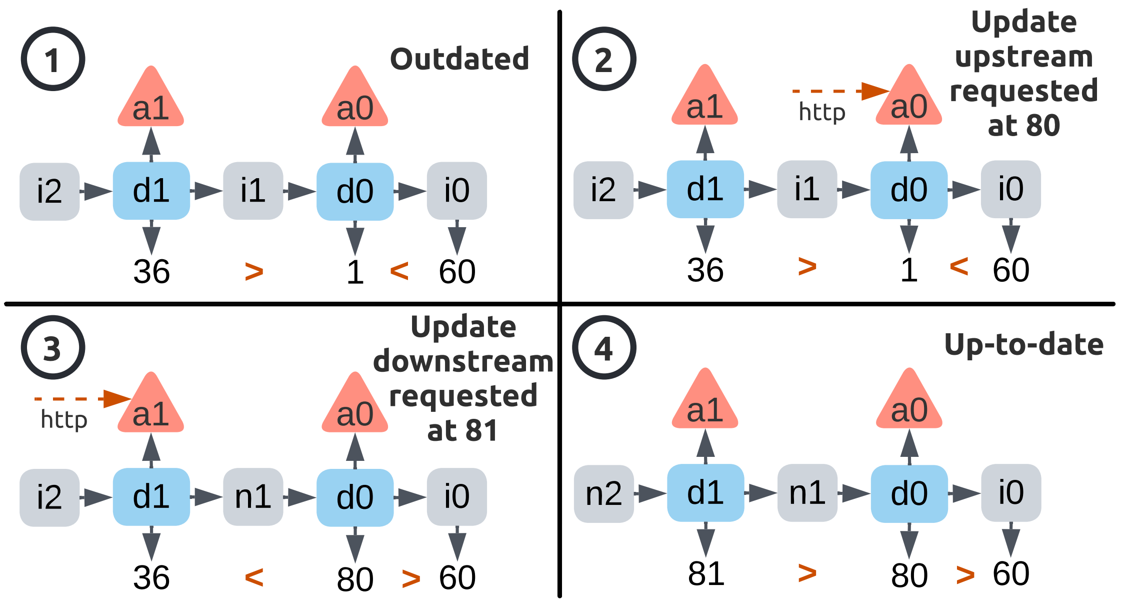


**Figure 9:** *The process of updating a derivation DAG consisting of only synchronous derivations.*

To expand this example to a slightly more complex situation, we consider accessing information *i2* in a linear chain of two synchronous derivations, as represented in Fig. 9. In this example, the input information of derivation *d1* belongs to derivation *d0*, therefore, the timeliness of *d1* is determined by comparing it with the derivation *d0*. However, just comparing the timestamps of these two will lead to an incorrect conclusion that *d1* is up-to-date. On the contrary, *d1* should be considered as outdated as it depends on an outdated derivation. Therefore, in the situation of assessing the timeliness of derivations that depend on derived information, the framework determines the timeliness of the upstream derivation (*d0*) first before performing any action to the downstream derivation (*d1*). For the presented example, assuming that information *i2* is accessed at time 80, the framework updates *d0* first, then *d1* immediately afterwards. With the new outputs connected in the knowledge graph, both derivations will be seen as up-to-date.

### 3.3.2 Purely asynchronous update

In the scientific computing domain for example, it is common to have situations where lengthy calculations from source input data are requested, requiring minutes or hours of wall time before the outputs are available. The asynchronous update suitable for such situations is discussed in this section, as described in Algorithm 2. It is similar to Algorithm 1, except that the algorithm for asynchronous derivations does not cache the derivation subgraph. Rather, the immediate upstream derivations are queried on-the-fly and hence their timeliness is determined purely based on real-time queries of the knowledge graph. The purpose of this design is to account for the fact that, due to the relatively long time scales involved, any information in the subgraph of asynchronous derivations can change while the process of carrying out an update is taking place.

**Algorithm 2:** updateAsyncDerivation(rootDerivationIRI)

```
Input: IRI of the root derivation instance
Result: The root derivation and all its upstream derivations are requested for update
        if deemed outdated
Create an empty directed acyclic graph $G$;
Query derivation instance $d$ from the KG using the given rootDerivationIRI;
updateAsyncDerivation($d$, $G$);

Function updateAsyncDerivation($d$, $G$):
    Query the list $U$ of all immediate upstream derivations of $d$;
    if $vertex\ d \notin G.vertices$ then
        Add $d$ as vertex in $G$;
    end
    for $u \in U$ do
        if $vertex\ u \notin G.vertices$ then
            $visited_u \leftarrow false$;
            Add $u$ as vertex in $G$;
        else
            $visited_u \leftarrow true$;
        end
        if $edge\ (d,u) \notin G.edges$ then
            $traversed_{d,u} \leftarrow false$;
            Add $(d,u)$ as edge in $G$ ;   /* will throw error if circular
             dependency detected */
        else
            $traversed_{d,u} \leftarrow true$;
        end
        if $visited_u == false$ and $traversed_{d,u} == false$ then
            updateAsyncDerivation($u$, $G$);
        end
    end
    Mark $d$ as Requested if it is deemed outdated;
```

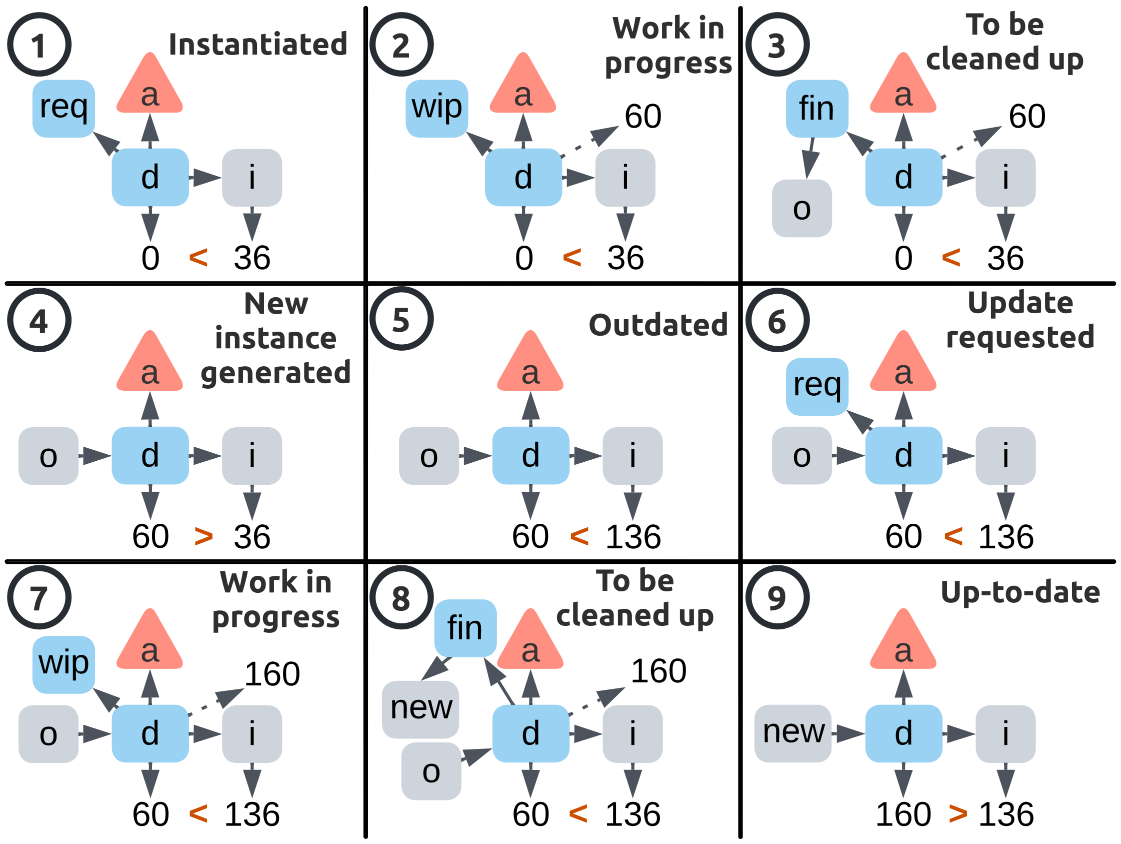


**Figure 10:** *The process of updating a single asynchronous derivation. The integers attached to the derivation instances via the dashed arrows denote the timestamps recorded by the data property* `retrievedInputsAt`.

As depicted by Fig. 10, we start from the point in time when the derivation is just instantiated, *i.e.* the asynchronous derivation is initialised with the status as `Requested` and a timestamp of 0, with no output computed. The actual update of an asynchronous derivation will be dealt with by the derivation agent and is not concurrent with the request for that update. As the agent periodically checks the status of derivations that are derived using itself, the requested derivation will be turned into `InProgress` at the next trigger and the timestamp when the inputs were read will be recorded. The successful completion of the job will be reflected in its status `Finished`. The agent will then connect the generated output to the derivation instance, update the timestamp and lastly remove the status altogether. The update process is similar to that of the initial computation. The only difference is that the agent will perform instance matching when reconnecting the newly derived information to existing downstream derivations in the derivation subgraph.

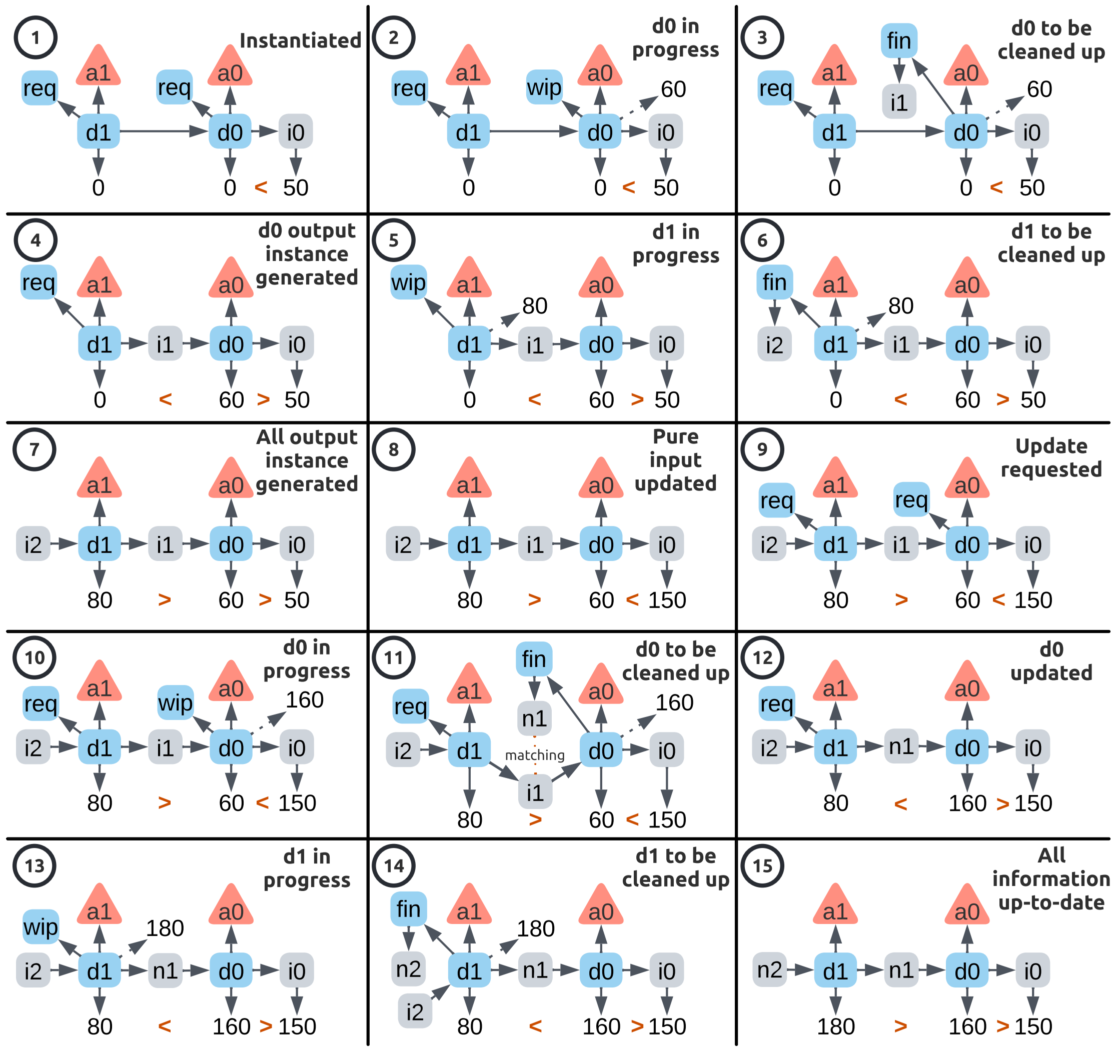


**Figure 11:** *The process of updating a derivation DAG consisting of only asynchronous derivations. The integers attached to the derivation instances via the dashed arrows denote the timestamps recorded by the data property* `retrievedInputsAt`.

The same example can be expanded to a linear chain of two asynchronous derivations instantiated with only one piece of input data, as illustrated in Fig. 11. In this case, as none of the outputs are computed, the downstream derivation *d1* is directly marked as `isDerivedFrom` its upstream derivation *d0*. The agent responsible for *d0* will operate in the same way as aforementioned, the only difference being that the agent will reconnect the new derived instance *i1* as input to derivation *d1* and remove the direct connection between the two derivations. Similar to scientific workflow, the agent that manages the downstream derivation *d1* can begin its execution only after its predecessors have finished successfully. Therefore, block 1 to 7 of Fig. 11 showcases one desired usage of the derivation framework to automatically complete a predefined workflow given input data. Once all derived instances are computed, we illustrate the steps for updating the derivations in block 8 to 15, where the source input data is updated. Upon request, the algorithm traverses the derivation chain and determines the timeliness of the derivations. Unlike synchronous update, the algorithm only marks derivations as `Requested`, leaving the actual update to individual agents in the same way as aforementioned.

### 3.3.3 Mixed-type update

The final example we provide is a derivation subgraph consisting of mixed-type derivations. Specifically, asynchronous derivations depend on synchronous derivations, *i.e.*, the lengthy calculation relies on results from fast computations. It is worth noting that the other way would lead to a purely asynchronous scenario. If a synchronous derivation is dependent on an asynchronous one, the synchronous derivation automatically becomes asynchronous due to the need to wait a long time for the response. For that reason, the case of asynchronous derivations depending on synchronous ones is the only mixed case that needs to be considered, without loss of generality.

As previously shown, we need to determine if a derivation is out-of-date and perform/request an update. These two parts are combined and executed in a recursive algorithm for derivation subgraphs that only consist of synchronous derivations. However, to save computation resources for updating mixed-type derivation subgraphs, we would like to design the framework to work in a way that the update of upstream synchronous derivations is only computed when the agent updates the asynchronous derivation. Therefore, when the algorithm recursively determines the timeliness of the asynchronous derivations, markup is needed in the knowledge graph to indicate that the downstream asynchronous derivation is in fact out-of-date and, hence, should be marked as `Requested`.

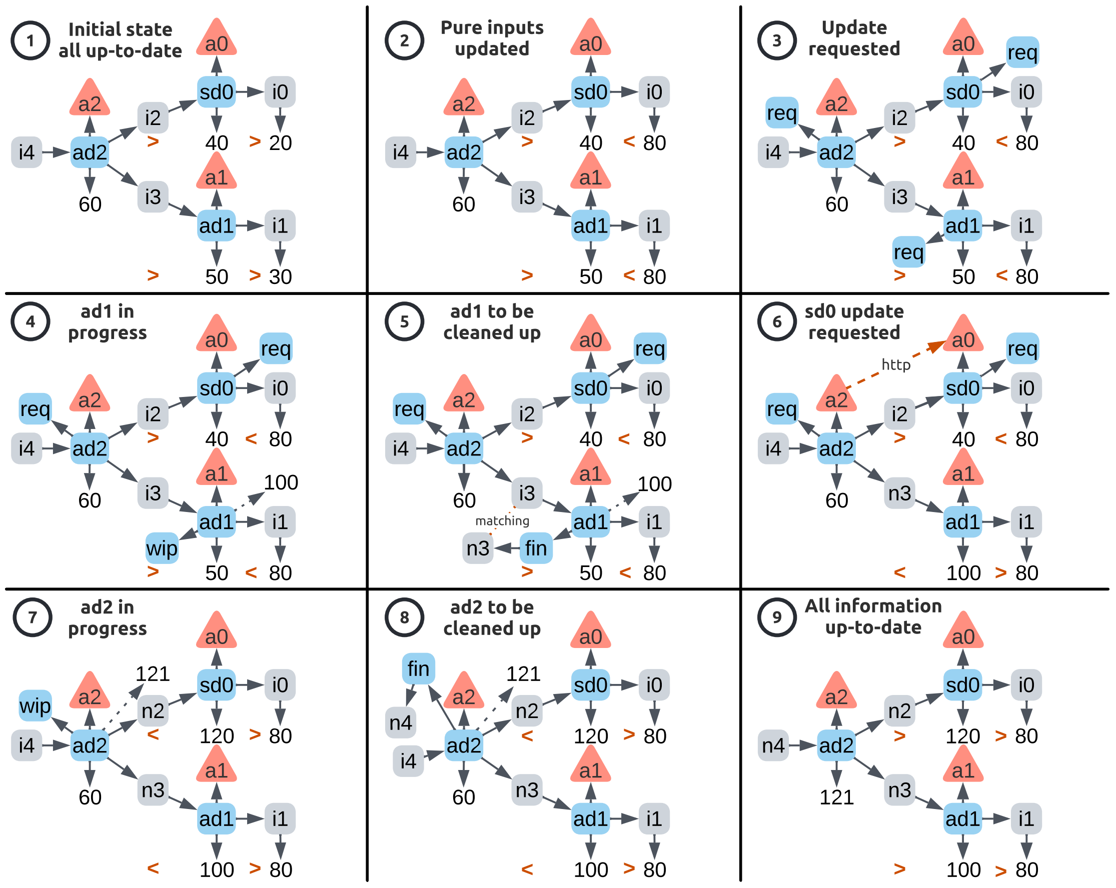


**Figure 12:** *The process of updating a derivation DAG consisting of asynchronous derivations that are dependent on synchronous derivations.*

A unified method is provided as a wrapper function of the two algorithms previously introduced. Depending on the instantiated type of root derivation, the function chooses a different entry algorithm. Figure 12 demonstrates the lifecycle of such an example. Synchronous derivation *sd0* will be marked as `Requested` when the algorithm traverses, which serves as a signal when the algorithm determines the timeliness of downstream asynchronous derivation *ad2*. The agent monitoring *ad2* will request an update for *sd0* and start its execution after it confirmed that all its immediate upstream asynchronous derivations are up-to-date.

So far we have introduced three parts for the derived information framework: the derivation ontology, the derivation agent, and the derivation client. These components work together in a cohesive fashion and we have included test cases to cover their key features and functionalities. There also exist minimal working examples in both Java and Python as tutorials for newcomers. For these resources, please refer to GitHub.

# 4 Use Case

In order to demonstrate the derivation framework, we apply it to evaluate the potential effects of a flooding event. The objective is to assess the number of buildings and the total property value that are potentially at risk. The details of the domain ontologies and agent logic can be found in [33]. Here, we focus on how the derivation framework is used for this UK-based use case with a simplified example, including the coverage of the derivation subgraph and the processes that occur as information is cascaded over time. The results for the actual data are presented using the Digital Twin Visualisation Framework (DTVF) that is part of the World Avatar.

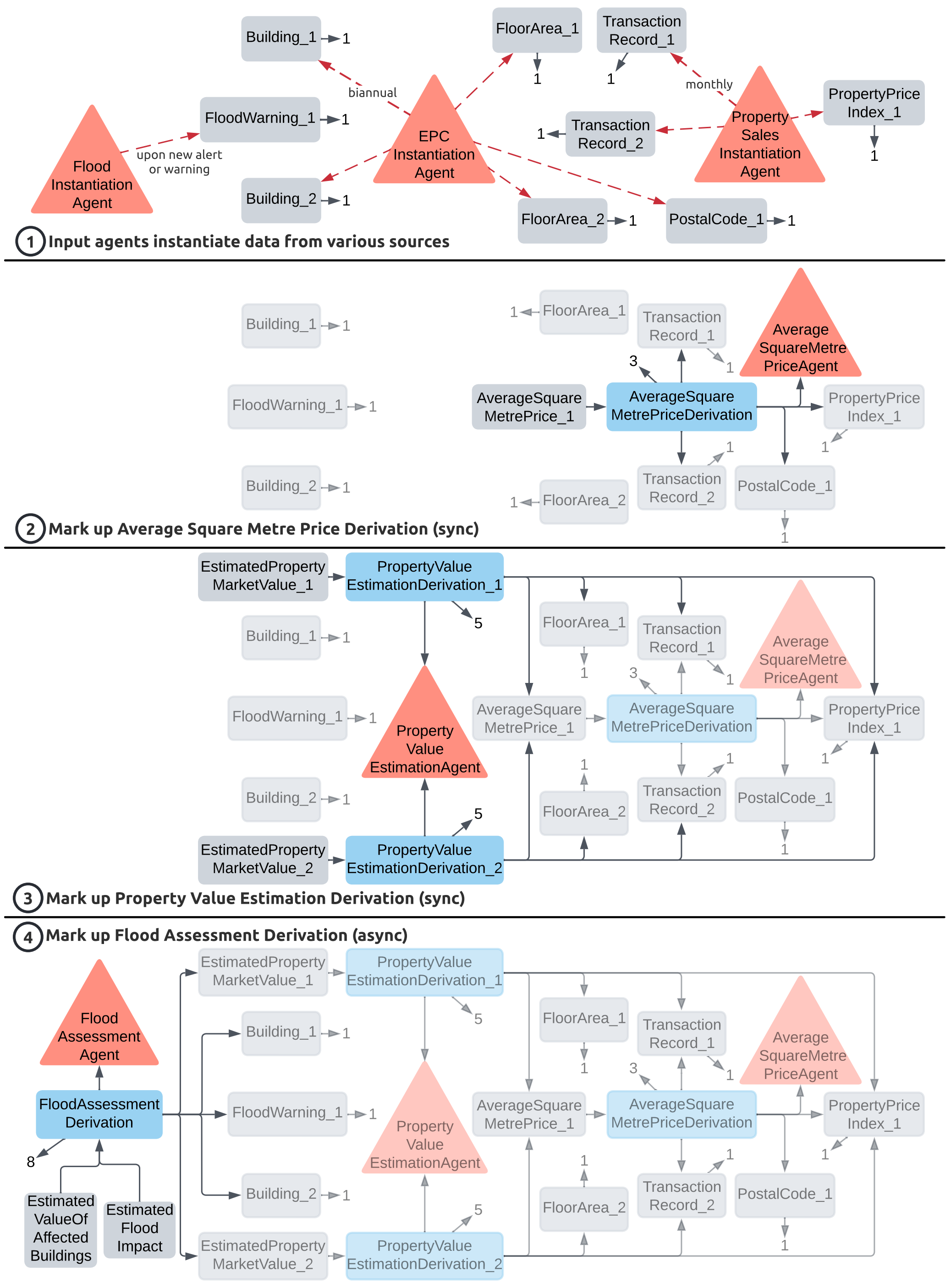
## 4.1 Automated population and update of the derivation subgraph

The flood impact assessment uses data from various sources, including application programming interfaces (APIs) such as the Environment Agency Real Time Flood-Monitoring API [20], Energy Performance Certificates (EPC) [21], and HM Land Registry Open Data [31]. These data are instantiated and updated in the World Avatar knowledge graph regularly by input agents. Using the source information, the impact assessment involves various derivation agents that work together to populate a derivation subgraph. This process encompasses two types of actions on the derivation subgraph: creating newly derived information, such as the impact of a newly issued flood warning, and updating existing information, such as the updated impact of an existing flood warning when some of the source information is updated.

One of these derivation agents, the Flood Assessment Agent, calculates the flood impact by identifying the buildings located within the affected area and determining the total market value of the properties at risk. The property value of each building is estimated by either scaling its most recent transaction record based on the local property price index or by multiplying its floor area by the average square metre price for its postal code. This representative average price can be computed by the Average Square Metre Price Agent considering all of the most recent transaction records in the area.

Figure 13 illustrates progresses in the derivation subgraph when evaluating the potential impact of an issued flood warning. As denoted by the red dashed arrow, the flood warning instance is instantiated by the Flood Instantiation Agent. It specifies information about the expected severity of a flood event and the specific geospatial extent that is at risk. In this simplified example, the geospatial polygon is assumed to cover a postal code that includes two buildings, each with data about its floor area and its most recent transaction record. Additionally, the UK property price index which captures changes in the value of residential properties is updated on a monthly basis.

Once the derivation agents are deployed, the population of derived information starts with marking up the average square metre price derivation for all postcodes within the region of interest. Next, the property value estimation derivation is marked up for all buildings. Since these computations are relatively fast, they are marked up as synchronous derivations to obtain instantaneous responses. The flood assessment derivation is marked up upon the instantiation of the flood warning instance. In practice, a flood warning can



Building_1
FloorArea_1
Transaction Record_1
biannual
monthly
PropertyPrice Index_1
FloodWarning_1
EPC Instantiation Agent
Property Sales Instantiation Agent
Transaction Record_2
Flood Instantiation Agent
upon new alert or warning
Building_2
FloorArea_2
PostalCode_1
1 Input agents instantiate data from various sources
Average SquareMetre PriceAgent
AverageSquare MetrePrice_1
AverageSquare MetrePriceDerivation
2 Mark up Average Square Metre Price Derivation (sync)
EstimatedProperty MarketValue_1
PropertyValue EstimationDerivation_1
Property Value EstimationAgent
EstimatedProperty MarketValue_2
PropertyValue EstimationDerivation_2
3 Mark up Property Value Estimation Derivation (sync)
4 Mark up Flood Assessment Derivation (async)
Flood Assessment Agent
FloodAssessment Derivation
Estimated ValueOf Affected Buildings
Estimated Flood Impact

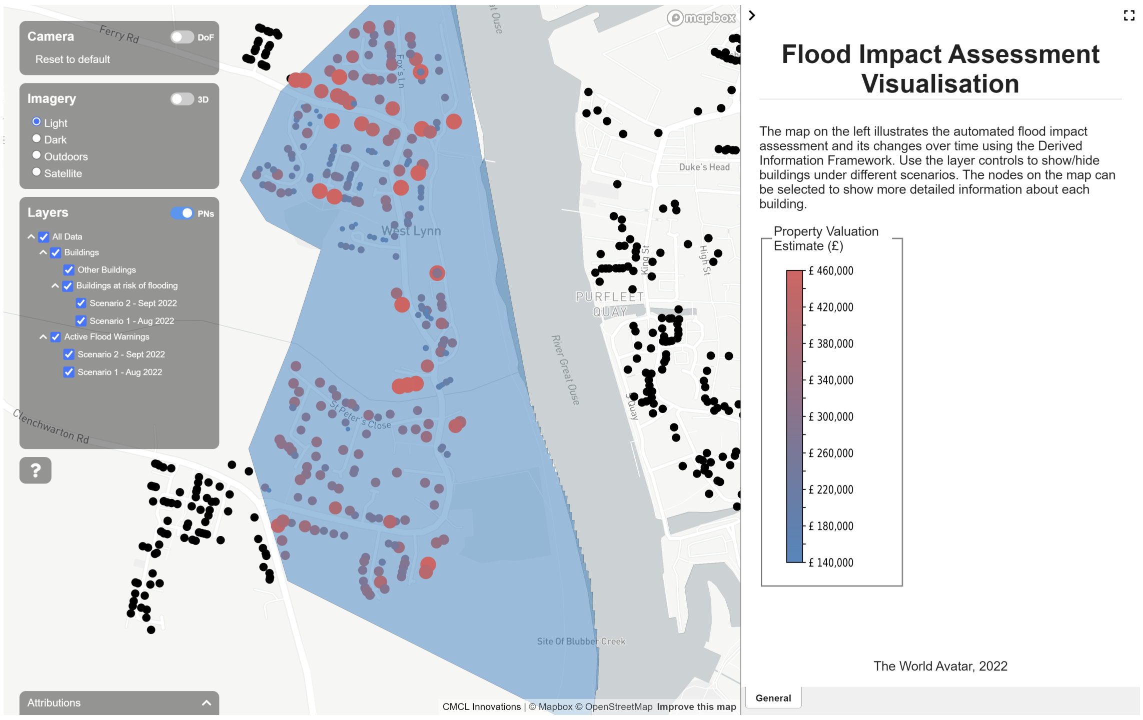
**Figure 13:** *The process of populating the derivation subgraph for flood impact assessment. Entities with low opacity represent existing entities in the knowledge graph, i.e. added in the previous steps of agents' operation before the agent adds newly derived information. The integers attached to the entities denote exemplary timestamps at which this information has been added to the knowledge graph.*

cover more than a dozen postal codes with hundreds of buildings. Its computation can take some time and is thus marked up as an asynchronous derivation.

As each input agent operates at different frequencies, the impact of a flood can alter when the source information is updated while the warning is still active. For example, the property price indices for all administrative districts in the UK are updated monthly, and the geospatial extent of an active flood warning can also change, both of which can result in outdated flood impact assessments. Upon an update request, the Flood Assessment Agent calls the Average Square Metre Price Agent and Property Value Estimation Agent in sequence to update the relevant derived information and creates an up-to-date flood impact estimate. Hence, this use case demonstrates both communication modes of the derived information framework by utilising both synchronous and asynchronous derivations.

## 4.2 Visualisation of potential flood impact

Following the simplified example, we present the visualisation of the impact assessment using real data. The derivation markup was created for a flood warning covering 34 postal codes and 289 buildings altogether.

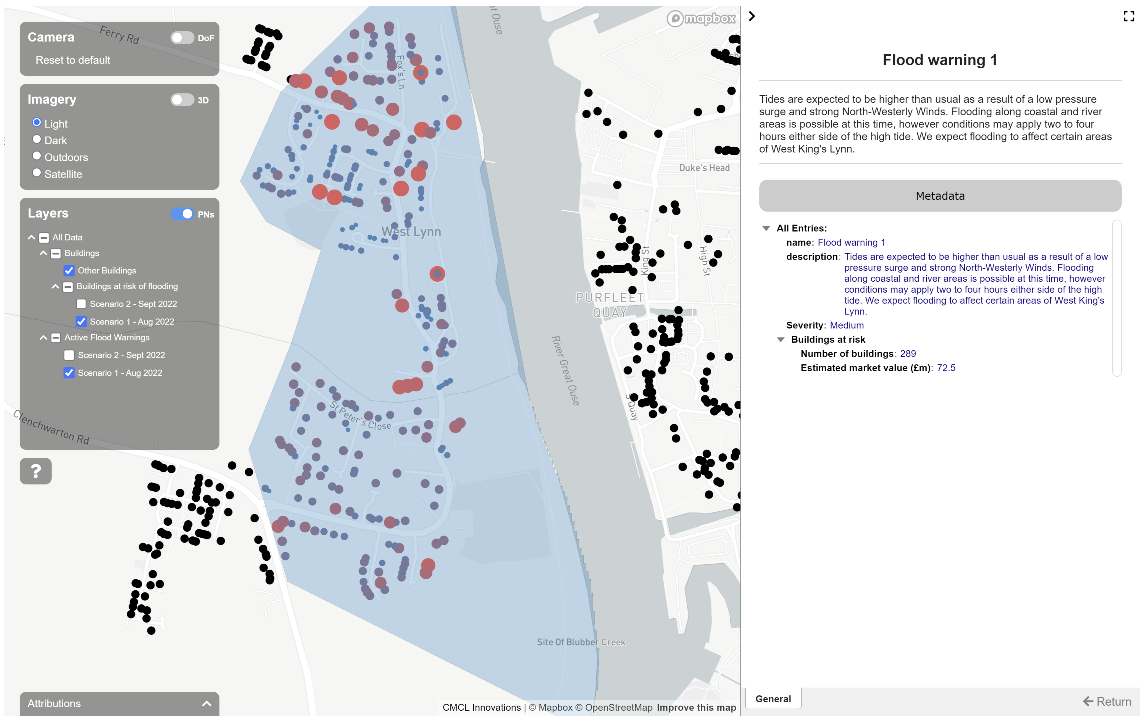

Flood Impact Assessment Visualisation

The map on the left illustrates the automated flood impact assessment and its changes over time using the Derived Information Framework. Use the layer controls to show/hide buildings under different scenarios. The nodes on the map can be selected to show more detailed information about each building.

**Figure 14:** *Web page visualising a flood impact assessment with layers for buildings and flood warnings. The blue region denotes a potential flooding area. The colour and size of the dots in this region reflect estimated property values. Buildings outside of the region are considered unaffected and are thus marked with black dots.*

Figure 14 visualises the estimated impact of a potential flooding event. The representation separates different data into distinct layers, including buildings and the geospatial area affected by the active flood warning. By overlaying the flood boundary on the map, it is possible to identify which properties are at risk. The buildings within the flooded region are colour-coded with their estimated property values, while the other buildings in the administrative district are included only with their location information.

Figure 15 presents two scenarios that are available to be selected on the plot: the estimated flood impact in August and September 2022. Between these two scenarios, an update of the district's property price index has been factored into calculating the total property value at risk. When clicking on the flooding area in different scenarios, the estimated impact and the detailed description of the flooding event are queried by the DTVF on-the-fly and displayed on the side panel. As the derivation agents manage the knowledge graph, the visualisation will be automatically updated.



Light [✓] · Dark · Outdoors · Satellite

Other Buildings ✓
Scenario 2 - Sept 2022
Scenario 1 - Aug 2022 ✓
Active Flood Warnings: Scenario 2 - Sept 2022; Scenario 1 - Aug 2022 ✓

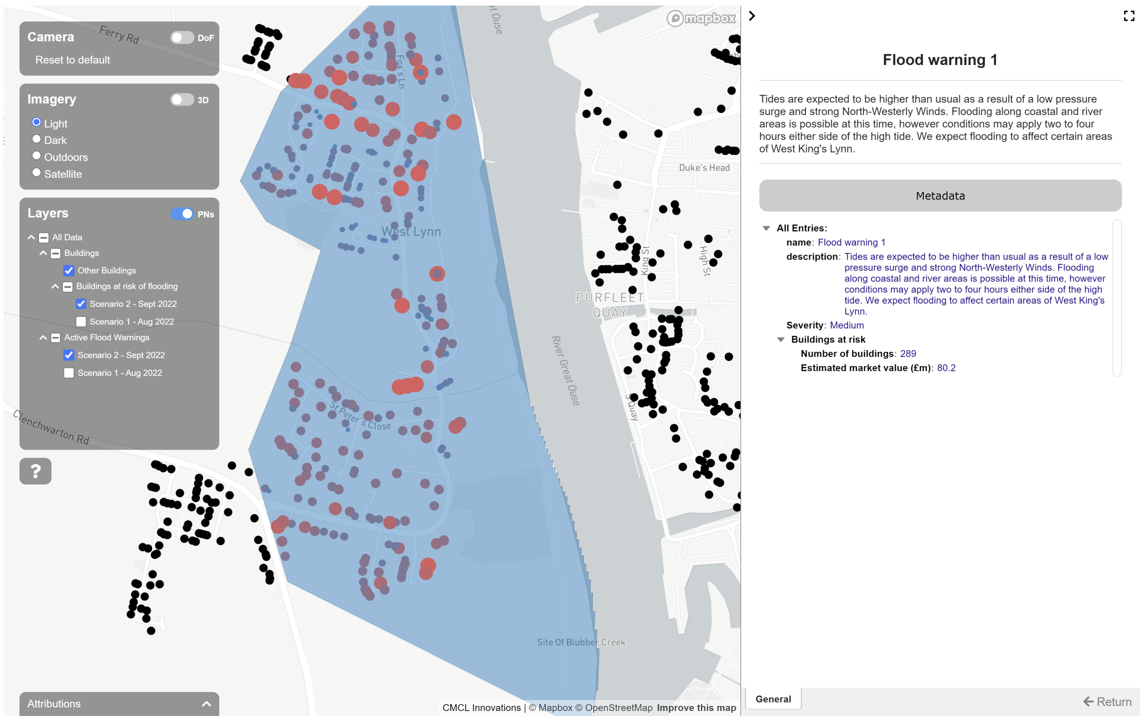
**Flood warning 1**

Tides are expected to be higher than usual as a result of a low pressure surge and strong North-Westerly Winds. Flooding along coastal and river areas is possible at this time, however conditions may apply two to four hours either side of the high tide. We expect flooding to affect certain areas of West King's Lynn.

Metadata

**All Entries:**
**name**: Flood warning 1
**description**: Tides are expected to be higher than usual as a result of a low pressure surge and strong North-Westerly Winds. Flooding along coastal and river areas is possible at this time, however conditions may apply two to four hours either side of the high tide. We expect flooding to affect certain areas of West King's Lynn.
**Severity**: Medium
**Buildings at risk**
**Number of buildings**: 289
**Estimated market value (£m)**: 72.5

(a) Scenario 1: Impact assessment of a newly issued flood warning.



Light [✓] · Dark · Outdoors · Satellite

Other Buildings ✓
Scenario 2 - Sept 2022 ✓
Scenario 1 - Aug 2022
Active Flood Warnings: Scenario 2 - Sept 2022 ✓; Scenario 1 - Aug 2022

**Flood warning 1**

Tides are expected to be higher than usual as a result of a low pressure surge and strong North-Westerly Winds. Flooding along coastal and river areas is possible at this time, however conditions may apply two to four hours either side of the high tide. We expect flooding to affect certain areas of West King's Lynn.

Metadata

**All Entries:**
**name**: Flood warning 1
**description**: Tides are expected to be higher than usual as a result of a low pressure surge and strong North-Westerly Winds. Flooding along coastal and river areas is possible at this time, however conditions may apply two to four hours either side of the high tide. We expect flooding to affect certain areas of West King's Lynn.
**Severity**: Medium
**Buildings at risk**
**Number of buildings**: 289
**Estimated market value (£m)**: 80.2

(b) Scenario 2: Impact update of an existing flood warning after property price index has increased.

**Figure 15:** *Automated flood impact assessment and update using the derived information framework. The side panel displays information about the clicked feature that has been dynamically retrieved from the knowledge graph.*

## 5 Conclusions

In this work, we developed a derived information framework as a knowledge-graph-native solution for tracking provenance and managing information within dynamic knowledge graphs. It expands previous capabilities and further abstracts complexity away from the developers of individual agents. The architecture includes a lightweight ontology for marking up agent communication as provenance records in the knowledge graph, an agent template that standardises the operation of agents in both synchronous and asynchronous communication modes, and a client library that offers functions for managing the derivation subgraph. The framework is technology-agnostic and is made available in both Java and Python.

To showcase the accessibility of the framework, it was applied to a flood impact assessment use case within the World Avatar project. The use case involves several derivation agents developed using the agent template. Once the source information is gathered by input agents from different APIs, the derivation subgraph is populated by creating derivation markups as requests for derivation agents to generate the derived information required in the impact assessment. If the input information is refreshed, the framework automatically updates the derived information when accessed. The results were visualised and can be deployed as a regular service if needed.

Future work includes expanding the framework to physical experimentation, implementing automated fault recovery for computations, evaluating the performance of different agents that can perform the same task, and incorporating automated service discovery for derivation markup generation.

## Research data

All the codes developed are available on The World Avatar GitHub repository https://github.com/cambridge-cares/TheWorldAvatar. The OntoDerivation TBox is available at file `JPS_Ontology/ontology/ontoderivation/OntoDerivation.owl`.

## Acknowledgements

This research was supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. Part of this work was supported by Towards Turing 2.0 under the EPSRC Grant EP/W037211/1 & The Alan Turing Institute. J. Bai acknowledges financial support provided by CSC Cambridge International Scholarship from Cambridge Trust and China Scholarship Council. M. Hofmeister acknowledges financial support provided by the Cambridge Trust and CMCL. M. Kraft gratefully acknowledges the support of the Alexander von Humboldt Foundation. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

# A Appendix

## A.1 Namespaces

om: <http://www.ontology-of-units-of-measure.org/resource/om-2/>
owl: <http://www.w3.org/2002/07/owl#>
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
rdfs: <http://www.w3.org/2000/01/rdf-schema#>
xsd: <http://www.w3.org/2001/XMLSchema#>
time: <http://www.w3.org/2006/time#>
OntoDerivation: <https://www.theworldavatar.com/kg/ontoderivation/>
OntoAgent: <http://www.theworldavatar.com/ontology/ontoagent/MSM.owl#>

## A.2 Description Logic representation of OntoDerivation

**Classes:**

$\text{Derivation} \sqsubseteq \top$
$\text{DerivationWithTimeSeries} \sqsubseteq \top$
$\text{DerivationAsyn} \sqsubseteq \top$
$\text{Status} \sqsubseteq \top$
$\text{Requested} \sqsubseteq \text{Status}$
$\text{InProgress} \sqsubseteq \text{Status}$
$\text{Finished} \sqsubseteq \text{Status}$
$\text{Error} \sqsubseteq \text{Status}$

**Object Properties:**

$\text{Derivation} \sqsubseteq \text{isDerivedFrom.owl:Thing}$
$\text{Derivation} \sqsubseteq \text{isDerivedUsing.OntoAgent:Service}$
$\text{DerivationWithTimeSeries} \sqsubseteq \text{isDerivedFrom.owl:Thing}$
$\text{DerivationWithTimeSeries} \sqsubseteq \text{isDerivedUsing.OntoAgent:Service}$
$\text{DerivationAsyn} \sqsubseteq \text{isDerivedFrom.owl:Thing}$
$\text{DerivationAsyn} \sqsubseteq \text{isDerivedUsing.OntoAgent:Service}$
$\text{DerivationAsyn} \sqsubseteq \text{hasStatus.Status}$
$\text{owl:Thing} \sqsubseteq \text{belongsTo.Derivation}$
$\text{owl:Thing} \sqsubseteq \text{belongsTo.DerivationWithTimeSeries}$
$\text{owl:Thing} \sqsubseteq \text{belongsTo.DerivationAsyn}$
$\text{Finished} \sqsubseteq \text{hasNewDerivedIRI.owl:Thing}$

**Data Properties:**

$\exists\, \text{retrievedInputsAt}.\top \sqsubseteq \text{DerivationAsyn}$
$\top \sqsubseteq \forall\, \text{retrievedInputsAt.xsd:Decimal}$

$\exists$ uuidLock.$\top \sqsubseteq$ DerivationAsyn
$\top \sqsubseteq \forall$ uuidLock.xsd:String

## A.3 Example queries

**Query 1:** *SPARQL query to obtain all derivation instances in the knowledge graph.*

```
PREFIX OntoDerivation: <https://www.theworldavatar.com/kg/ontoderivation/>
PREFIX time: <http://www.w3.org/2006/time#>

SELECT ?derivation ?devTime ?inputTime ?status ?status_type
WHERE {
  VALUES ?derivationType {
    OntoDerivation:DerivationAsyn
    OntoDerivation:Derivation
    OntoDerivation:DerivationWithTimeSeries
  }
  ?derivation a ?derivationType;
  time:hasTime/time:inTimePosition/time:numericPosition ?devTime.
  OPTIONAL {
    ?derivation OntoDerivation:isDerivedFrom ?upstream.
    ?upstream time:hasTime/time:inTimePosition/time:numericPosition ?inputTime.
  }
  OPTIONAL {
    ?derivation OntoDerivation:hasStatus ?status.
    ?status a ?status_type.
  }
}
```

**Query 2:** *SPARQL query to map the derivation inputs to agent I/O signature.*

```
PREFIX OntoDerivation: <https://www.theworldavatar.com/kg/ontoderivation/>
PREFIX OntoAgent: <http://www.theworldavatar.com/ontology/ontoagent/MSM.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?input ?type
WHERE {
  <agentIRI> OntoAgent:hasOperation/OntoAgent:hasInput/
    OntoAgent:hasMandatoryPart/OntoAgent:hasType ?type .
  <derivationIRI> OntoDerivation:isDerivedFrom ?input .
  ?input a*/rdfs:subClassOf* ?type .
}
```

**Query 3:** *SPARQL query to determine the immediate upstream derivations that requires an update.*

```
PREFIX OntoDerivation: <https://www.theworldavatar.com/kg/ontoderivation/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?upstreamDerivation ?upstreamDerivationType
WHERE {
  {
    <derivationIRI> OntoDerivation:isDerivedFrom/
      OntoDerivation:belongsTo? ?upstreamDerivation .
    ?upstreamDerivation a ?upstreamDerivationType .
    VALUES ?upstreamDerivationType {
      OntoDerivation:Derivation
      OntoDerivation:DerivationWithTimeSeries
      OntoDerivation:DerivationAsyn
    }
  }
  ?upstreamDerivation time:hasTime/time:inTimePosition/
    time:numericPosition ?upstreamDerivationTimestamp .
  OPTIONAL {
    ?upstreamDerivation OntoDerivation:hasStatus ?status .
    {
      ?status a ?statusType .
      FILTER (?statusType != owl:Thing && ?statusType != owl:NamedIndividual)
     }
  }
  OPTIONAL {
    ?upstreamDerivation OntoDerivation:isDerivedFrom/time:hasTime/
      time:inTimePosition/time:numericPosition ?pureInputTimestamp .
  }
  OPTIONAL {
    ?upstreamDerivation OntoDerivation:isDerivedFrom/OntoDerivation:belongsTo/
      time:hasTime/time:inTimePosition/time:numericPosition
      ?inputsBelongingToDerivationTimestamp .
  }
  FILTER (?upstreamDerivationTimestamp < ?pureInputTimestamp ||
    ?upstreamDerivationTimestamp < ?inputsBelongingToDerivationTimestamp ||
    ?statusType = OntoDerivation:Requested ||
    ?statusType = OntoDerivation:InProgress ||
    ?statusType = OntoDerivation:Finished ||
    ?statusType = OntoDerivation:Error)
}
```

**Query 4:** *SPARQL query to update the derivation in the knowledge graph.*

```
PREFIX OntoDerivation: <https://www.theworldavatar.com/kg/ontoderivation/>
PREFIX time: <http://www.w3.org/2006/time#>

DELETE {
  ?e ?p1 ?o .
  ?s ?p2 ?e .
  ?d OntoDerivation:hasStatus ?status .
  ?status a ?statusType .
  ?timeIRI time:numericPosition ?dTs .
}
INSERT {
  <newInstance1> a <rdfTypeOfNewInstance1> .
  <newInstance2> a <rdfTypeOfNewInstance2> .
  <newInstance3> a <rdfTypeOfNewInstance3> .
  <newInstance3> OntoDerivation:belongsTo <derivationIRI> .
  <newInstance1> OntoDerivation:belongsTo <derivationIRI> .
  <downstreamDerivation1> OntoDerivation:isDerivedFrom <newInstance1> .
  <downstreamDerivation2> OntoDerivation:isDerivedFrom <newInstance1> .
  <newInstance2> OntoDerivation:belongsTo <derivationIRI> .
  <downstreamDerivation3> OntoDerivation:isDerivedFrom <newInstance2> .
  ?timeIRI time:numericPosition 1659981343 .
}
WHERE {
  {
    SELECT ?d ?timeIRI ?dTs ?status ?statusType ?e ?p1 ?o ?s ?p2
    WHERE {
      {
        VALUES ?d {<derivationIRI> }
        ?d time:hasTime/time:inTimePosition ?timeIRI .
        ?timeIRI time:numericPosition ?dTs .
        ?d OntoDerivation:isDerivedFrom/OntoDerivation:belongsTo? ?ups .
        ?ups time:hasTime/time:inTimePosition/time:numericPosition ?upsTs .
        FILTER (?dTs < ?upsTs)
      }
      {
        ?e OntoDerivation:belongsTo ?d .
        ?e ?p1 ?o .
        OPTIONAL { ?s ?p2 ?e . }
      }
      OPTIONAL {
        ?d OntoDerivation:hasStatus ?status .
        ?status a ?statusType .
      }
    }
  }
}
```

## References

[1] J. Akroyd, S. Mosbach, A. Bhave, and M. Kraft. Universal Digital Twin – A Dynamic Knowledge Graph. *Data-Centric Engineering*, 2:e14, 2021. doi:10.1017/dce.2021.10.

[2] J. Akroyd, A. Bhave, G. Brownbridge, E. Christou, M. Hillman, M. Hofmeister, M. Kraft, J. Lai, K. F. Lee, S. Mosbach, D. Nurkowski, and O. Parry. CReDo Technical Paper 1: Building a Cross-Sector Digital Twin, 2022. URL https://digitaltwinhub.co.uk/credo/technical/1-building-a-cross-sector-twin/. Accessed 19 July 2022.

[3] J. Akroyd, Z. Harper, D. Soutar, F. Farazi, A. Bhave, S. Mosbach, and M. Kraft. Universal Digital Twin: Land Use. *Data-Centric Engineering*, 3, 2022. doi:10.1017/dce.2021.21.

[4] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *Proceedings of 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 423–424. IEEE, 2004. doi:10.1109/SSDM.2004.1311241.

[5] J. Bai, R. Geeson, F. Farazi, S. Mosbach, J. Akroyd, E. J. Bringley, and M. Kraft. Automated Calibration of a Poly(oxymethylene) Dimethyl Ether Oxidation Mechanism Using the Knowledge Graph Technology. *J. Chem. Inf. Model.*, 61(4):1701–1717, 2021. doi:10.1021/acs.jcim.0c01322.

[6] J. Bai, L. Cao, S. Mosbach, J. Akroyd, A. A. Lapkin, and M. Kraft. From Platform to Knowledge Graph: Evolution of Laboratory Automation. *JACS Au*, 2(2):292–309, 2022. doi:10.1021/jacsau.1c00438.

[7] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Sci. Am.*, 284(5): 34–43, 2001. doi:10.1038/scientificamerican0501-34. URL https://www.jstor.org/stable/10.2307/26059207.

[8] blazegraph. Reification Done Right, 2020. URL https://github.com/blazegraph/database/wiki/Reification_Done_Right.

[9] A. Chadzynski, N. Krdzavac, F. Farazi, M. Q. Lim, S. Li, A. Grisiute, P. Herthogs, A. von Richthofen, S. Cairns, and M. Kraft. Semantic 3D City Database - An Enabler for a Dynamic Geospatial Knowledge Graph. *Energy and AI*, 6:100106, 2021. doi:10.1016/j.egyai.2021.100106.

[10] P. Ciccarese, S. Soiland-Reyes, K. Belhajjame, A. J. G. Gray, C. Goble, and T. Clark. PAV Ontology: Provenance, Authoring and Versioning. *J. Biomed. Semant.*, 4(1): 37, 2013. doi:10.1186/2041-1480-4-37. URL https://pav-ontology.github.io/pav/.

[11] S. Cox, C. Little, J. R. Hobbs, and F. Pan. Time Ontology in OWL. W3C Candidate Recommendation 26 March 2020, 2020. URL https://www.w3.org/TR/owl-time/. Accessed 21 July 2022.

[12] R. F. da Silva, H. Casanova, K. Chard, D. Laney, D. Ahn, S. Jha, C. Goble, L. Ramakrishnan, L. Peterson, B. Enders, D. Thain, I. Altintas, Y. Babuji, R. M. Badia, V. Bonazzi, T. Coleman, M. Crusoe, E. Deelman, F. D. Natale, P. D. Tommaso, T. Fahringer, R. Filgueira, G. Fursin, A. Ganose, B. Gruning, D. S. Katz, O. Kuchar, A. Kupresanin, B. Ludascher, K. Maheshwari, M. Mattoso, K. Mehta, T. Munson, J. Ozik, T. Peterka, L. Pottier, T. Randles, S. Soiland-Reyes, B. Tovar, M. Turilli, T. Uram, K. Vahi, M. Wilde, M. Wolf, and J. Wozniak. Workflows Community Summit: Bringing the Scientific Workflows Community Together, 2021. URL `https://arxiv.org/abs/2103.09181`. Accessed 18 July 2022.

[13] Dapr Authors. APIs for Building Portable and Reliable Microservices, 2022. URL `https://dapr.io/`. Accessed 14 July 2022.

[14] DCMI Usage Board. Bibliographic Ontology (BIBO) in RDF, 2016. URL `https://www.dublincore.org/specifications/bibo/bibo/`.

[15] DCMI Usage Board. DCMI Metadata Terms, 2020. URL `https://www.dublincore.org/specifications/dublin-core/dcmi-terms/`.

[16] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Sci. Program.*, 13(3):219–237, 2005. doi:10.1155/2005/128026.

[17] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An Overview of Workflow System Features and Capabilities. *Future Gener. Comput. Syst.*, 25(5):528–540, 2009. doi:10.1016/j.future.2008.06.012.

[18] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny, and K. Wenger. Pegasus, A Workflow Management System for Science Automation. *Future Gener. Comput. Syst.*, 46:17–35, 2015. doi:10.1016/j.future.2014.10.008.

[19] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter. The Future of Scientific Workflows. *The International Journal of High Performance Computing Applications*, 32(1):159–175, 2018. doi:10.1177/1094342017704893.

[20] Department for Environment Food & Rural Affairs. Real Time flood-monitoring API, 2021. URL `https://environment.data.gov.uk/flood-monitoring/doc/reference`. Accessed 4 Feb 2022.

[21] Department for Levelling Up, Housing & Communities. Energy Performance of Buildings Data, 2022. URL `https://epc.opendatacommunities.org/docs/api`. Accessed 24 Feb 2022.

[22] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame. Nextflow Enables Reproducible Computational Workflows. *Nat. Biotechnol.*, 35(4):316–319, 2017. doi:10.1038/nbt.3820.

[23] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. Microservices: Yesterday, Today, and Tomorrow. *Present and Ulterior Software Engineering*, pages 195–216, 2017. doi:10.1007/978-3-319-67425-4_12.

[24] W. Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019. URL `https://github.com/Lightning-AI/lightning`. Accessed 14 July 2022.

[25] F. Farazi, J. Akroyd, S. Mosbach, P. Buerger, D. Nurkowski, M. Salamanca, and M. Kraft. OntoKin: An Ontology for Chemical Kinetic Reaction Mechanisms. *J. Chem. Inf. Model.*, 60(1):108–120, 2020. doi:10.1021/acs.jcim.9b00960.

[26] F. Farazi, N. B. Krdzavac, J. Akroyd, S. Mosbach, A. Menon, D. Nurkowski, and M. Kraft. Linking Reaction Mechanisms and Quantum Chemistry: An Ontological Approach. *Comput. Chem. Eng.*, 137:106813, 2020. doi:10.1016/j.compchemeng.2020.106813.

[27] C. Gutierrez and J. F. Sequeda. Knowledge Graphs. *Commun. ACM*, 64(3):96–104, 2021. doi:10.1145/3418294.

[28] O. Hartig and B. Thompson. Foundations of an Alternative Approach to Reification in RDF, 2021. URL `https://arxiv.org/abs/1406.3399v3`.

[29] J. Hendler. Agents and the Semantic Web. *IEEE Intell. Syst.*, 16(2):30–37, 2001. doi:10.1109/5254.920597.

[30] P. Hitzler. A Review of The Semantic Web Field. *Commun. ACM*, 64(2):76–83, 2021. doi:10.1145/3397512.

[31] HM Land Registry. HM Land Registry Open Data, 2022. URL `https://landregistry.data.gov.uk/`. Accessed 13 Oct 2022.

[32] M. Hofmeister, S. Mosbach, J. Hammacher, M. Blum, G. Röhrig, C. Dörr, V. Flegel, A. Bhave, and M. Kraft. Resource-Optimised Generation Dispatch Strategy for District Heating Systems Using Dynamic Hierarchical Optimisation. *Appl. Energy*, 305:117877, 2022. doi:10.1016/j.apenergy.2021.117877.

[33] M. Hofmeister, G. Brownbridge, J. Akroyd, S. Mosbach, M. Hillman, J. Bai, F. Farazi, A. Chadzynski, and M. Kraft. Universal digital twin – cross-domain flood assessment in smart cities using knowledge graphs, 2023. Submitted for publication.

[34] A. Hogan, E. Blomqvist, M. Cochez, C. D'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann. Knowledge Graphs. *ACM Comput. Surv.*, 54(4):1–37, 2022. doi:10.1145/3447772.

[35] M. Krämer, H. M. Würz, and C. Altenhofen. Executing Cyclic Scientific Workflows in the Cloud. *J. Cloud Comput.*, 10(1):1–26, 2021. doi:10.1186/s13677-021-00229-7.

[36] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao. PROV-O: The PROV Ontology. W3C Recommendation, 2013. URL `https://www.w3.org/TR/prov-o/`.

[37] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia – A Large-Scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015. doi:10.3233/SW-140134.

[38] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. A Survey of Data-Intensive Scientific Workflow Management. *J. Grid Comput.*, 13(4):457–493, 2015. doi:10.1007/s10723-015-9329-8.

[39] N. Lopes, A. Zimmermann, A. Hogan, G. Lukácsy, A. Polleres, U. Straccia, and S. Decker. RDF Needs Annotations. W3C Workshop – RDF Next Steps, Stanford, CA, USA, June 26-27, 2010. URL `https://www.w3.org/2009/12/rdf-ws/papers/ws09`.

[40] E. Lyons, G. Papadimitriou, C. Wang, K. Thareja, P. Ruth, J. Villalobos, I. Rodero, E. Deelman, M. Zink, and A. Mandal. Toward a Dynamic Network-Centric Distributed Cloud Platform for Scientific Workflows: A Case Study for Adaptive Weather Sensing. In *2019 15th International Conference on eScience (eScience)*, pages 67–76. IEEE, 2019. doi:10.1109/eScience.2019.00015.

[41] L. Moreau, L. Ding, J. Futrelle, D. G. Verdejo, P. Groth, M. Jewell, S. Miles, P. Missier, J. Pan, and J. Zhao. Open Provenance Model (OPM) OWL Specification, 2010. URL `https://openprovenance.org/opm/model/opmo`.

[42] S. Mosbach, A. Menon, F. Farazi, N. Krdzavac, X. Zhou, J. Akroyd, and M. Kraft. Multiscale Cross-Domain Thermochemical Knowledge-Graph. *J. Chem. Inf. Model.*, 60(12):6155–6166, 2020. doi:10.1021/acs.jcim.0c01145.

[43] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. Industry-Scale Knowledge Graphs: Lessons and Challenges. *Commun. ACM*, 62(8):36–43, 2019. doi:10.1145/3331166.

[44] ontotext. What is RDF-star?, 2022. URL `https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-star/`.

[45] T. Pellissier Tanon, D. Vrandečić, S. Schaffert, T. Steiner, and L. Pintscher. From Freebase to Wikidata: The Great Migration. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1419–1428, 2016. doi:10.1145/2872427.2874809.

[46] T. Savage, J. Akroyd, S. Mosbach, M. Hillman, F. Sielker, and M. Kraft. Universal Digital Twin–The Impact of Heat Pumps on Social Inequality. *Adv. Appl. Energy*, 5: 100079, 2022. doi:10.1016/j.adapen.2021.100079.

[47] L. F. Sikos and D. Philp. Provenance-Aware Knowledge Representation: A Survey of Data Models and Contextualized Knowledge Graphs. *Data Sci. and Eng.*, 5(3): 293–316, 2020. doi:10.1007/s41019-020-00118-0.

[48] The Apache Software Foundation. Apache Airflow, 2022. URL https://airflow.apache.org/. Accessed 17 July 2022.

[49] World Wide Web Consortium (W3C). RDF-star, 2021. URL https://w3c.github.io/rdf-star/.

[50] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *J. Grid Comput.*, 3(3):171–200, 2005. doi:10.1007/s10723-005-9010-8.

[51] J. Zhao, C. Bizer, Y. Gil, P. Missier, and S. Sahoo. Provenance Requirements for the Next Version of RDF. W3C Workshop – RDF Next Steps, Stanford, CA, USA, June 26-27, 2010. URL https://www.w3.org/2009/12/rdf-ws/papers/ws08.

[52] X. Zhou, A. Eibeck, M. Q. Lim, N. B. Krdzavac, and M. Kraft. An Agent Composition Framework for the J-Park Simulator - A Knowledge Graph for the Process Industry. *Comput. Chem. Eng.*, 130:106577, 2019. doi:10.1016/j.compchemeng.2019.106577.