UNIVERSITÁ DI PISA
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

DATA FLOW QUALITY MONITORING IN DATA INFRASTRUCTURES

DOCTORAL THESIS

Author
Andrea Mannocci

Tutor (s)
Prof. Marco Avvenuti, Dr. Paolo Manghi

Reviewer (s)
Prof. Andreas Rauber, Marta Mattoso

The Coordinator of the PhD Program
Prof. Marco Luise

Pisa, November 2016

XXIX

To the ones who believed in me and always will…
maybe even more than I am used to.

"Among all the sure things,
the most certain is doubt."
— Bertolt Brecht

Acknowledgements

PURSUING this Ph.D. and putting this manuscript together have been not only my last two labours, which several times led me to the brink of an emotional outburst, but also a transition phase in my life. New and old challenges, experiences, jobs, countries, cities, houses, loves and friends. Oh boy, what a hectic period! However, I apparently survived it and I arrived to peek the light at the end of the tunnel, so with a burst of joy I'd like to thank all the people that made this possible.

My first and foremost "thank you" goes to Dr. Paolo Manghi whose ideas, suggestions and guidance steered and inspired me during this three year-long journey in this valley of satisfaction and despair called Ph.D. He contributed to this work more than anyone with both exciting and passionate discussions, so I owe him my profound gratitude. I wish to thank also Prof. Marco Avvenuti who welcomed the supervision of this Ph.D. project and Prof. Andreas Rauber and Prof. Marta Mattoso for reviewing this manuscript and providing me with such constructive feedback.

I'd like to extend my gratitude to all the members of the NeMIS laboratory at ISTI-CNR: it has been, both metaphorically and literally, my second home for these last four years. Here I've had the opportunity to challenge myself and let my ideas thrive. My gratitude also goes to my "roommates", whose company made days pass by working and studying, but also laughing and chatting. You made my average day always pleasant, so here it is... my deepest thanks go to you Sandro La Bruzzo, Claudio Atzori, Michele Artini, Alessia Bardi and Miriam Baglioni. Thanks also to Andrea Dell'Amico and Marko Mikulicic with whom I shared, despite not being in the same room, thoughts and concerns about monitoring in all its shapes, and to Monja Da Riva, the only shiatsu-techno therapist I know, for all the laughter and for sharing sesame snacks cravings with me; best of luck for your career. An incommensurable thank you goes to Dr. Leonardo Candela for endless talks about data infrastructures and papers, and to Alessandro Nardi and Catherine Bosio for their terrific back-office work and for sharing their life experiences and suggestions with me. Last but not least, I'd like to thank Dr. Donatella Castelli whose experience and humanity created such an amazing and active group, and the institute which gave me the opportunity to apply for a visiting period at The Open University; thing that leads to the next series of acknowledgements.

Deep thanks to the Knowledge Media Institute for having me as a visiting Ph.D. student and especially to Dr. Peter Knoth and the CORE team: Matteo, Lucas, Sam, Dasha, Nancy, Aris and Váša. Thanks to the rest of the KMi gang, the Knowledge Media instruments band and the "OU bubble": thank you Aitor, Alberto, the two Alessandro, Allan, Angelo, Beppe, Bjarte, Brian, Davide, Dinar, Prof. Enrico Motta, Ginevra, Giorgio, Ilaria, Jay, Julian, Koula, Lara (who, at the time of writing, counts for two), Maria, Martin, Manu, Michael, Miriam, Paco, Patrizia, Simon and Tina. I hope I didn't forget anyone, in which case I beg your pardon. My gratitude also goes to Dr. Mathieu D'Aquin for offering me a tremendous opportunity as a PostDoc at KMi and to the KMI-er Enrico Daga, whose life choices and studies brought him to be a workflows and guitars expert and thus, poor him, so damn interesting to my eyes. Thanks for countless minutes spent chatting.

A special mention goes to Penelope for being there everyday listening to me, probably without having the faintest clue, talking about this thesis and providing me with useful suggestions. Your love and patience – *in nomen omen* said the ancient Romans – largely contributed in keeping my sanity at bay and won't be forgotten.

Let me address my deepest gratitude to my parents, my lone and only Family; it's not the quantity but the quality that matters. I've mentioned it already in previous dedications, but I want to stress it again here: thanks for educating me in the respect of both science and literature; your support and unconditional love always pushes me in pursuing my own interests and do my best. It's been an honor to make you proud.

Thanks to Niccolò, Luigi, Flavio, Ilaria and all my old and new friends for countless nights out and good time, to Monse for being a remote anchor point throughout all these years, to the rest of the DpNn clan which amused me online in my little spare time, to Livia who relentlessly tried to pull the best out of me, to Valentina for our long chats, and to Rossella and Ludovica for all the silly videos that cheered my days up.

Somebody once made me notice that our life and our present self is the summation of infinitesimal events and choices aligned in a timeline in which every instant, every circumstantial action, even the most random encounter, takes part to the outcome. Even if in engineering infinitesimal terms are reasonably disregarded somewhere along the way, I'd like to extend my gratitude to everyone quoting two songs dear to me.

Here's to us,
here's to love,
all the times
that we messed up.
Here's to you,
fill the glass

(Halestorm – "Here's to us")

Now that we're here,
it's so far away,
all the struggle we thought was in vain
and all the mistakes
one life contained
they all finally start to go away.
Now that we're here,
it's so far away,
and I feel like I can face the day,
and I can forgive and I'm not ashamed
to be the person that I am today.

(Staind – "So far away")

Abstract

In the last decade, a lot of attention worldwide has been brought by researchers, organizations, and funders on the realization of *Data Infrastructures (DIs)*, namely systems supporting researchers with the broad spectrum of resources they need to perform science. DIs are here intended as ICT (eco)systems offering *data* and *processing components* which can be combined into *data flows* so as to enable arbitrarily complex data manipulation actions serving the consumption needs of DI customers, be them humans or machines.

Data resulting from the execution of data flows, represent an important asset both for the DI users, typically craving for the information they need, and for the organization (or community) operating the DI, whose existence and cost sustainability depends on the adoption and usefulness of the DI. On the other hand, when operating several data processing data flows over time, several issues, well-known to practitioners, may arise and compromise the behaviour of the DI, and therefore undermine its reliability and generate stakeholders dissatisfaction. Such issues span a plethora of causes, such as (i) the lack of any kind of guarantees (e.g. quality, stability, findability, etc.) from integrated external data sources, typically not under the jurisdiction of the DI; (ii) the occurrence at any abstraction level of subtle, unexpected errors in the data flows; and (iii) the nature in ever changing evolution of the DI, in terms of data flow composition and algorithms/configurations in use.

The autonomy of DI components, their use across several data flows, the evolution of end-user requirements over time, make the one of DI data flows a critical environment, subject to the most subtle inconsistencies. Accordingly, DI users demand guarantees, while *quality managers* are called to provide them, on the "correctness" of the DI data flows behaviour over time, to be somehow quantified in terms of "data quality" and in terms of "processing quality". Monitoring the quality of data flows is therefore a key activity of paramount importance to ensure the up-taking and long term existence of a DI. Indeed, monitoring can detect or anticipate misbehaviours of DI's data flows, in order to prevent and adjust the errors, or at least "formally" justify to the stakeholders the underlying reasons, possibly not due to the DI, of such errors. Not only, monitoring can also be vital for DIs operation, as having hardware and software resources actively

employed in processing low quality data can yield inefficient resource allocation and waste of time.

However, data flow quality monitoring is further hindered by the "hybrid" nature of such infrastructures, which typically consist of a *patchwork* of individual components ("system of systems") possibly developed by distinct stakeholders with possibly distinct life-cycles, evolving over time, whose interactions are regulated mainly by shared policies agreed at infrastructural level. Due to such heterogeneity, generally DIs are not equipped with built-in monitoring systems in this sense and to date DI quality managers are therefore bound to use combinations of existing tools – with non trivial integration efforts – or to develop and integrate *ex-post* their own *ad-hoc* solutions, at high cost of realization and maintenance.

In this thesis, we introduce MoniQ, a general-purpose Data Flow Quality Monitoring system enabling the monitoring of critical data flow components, which are routinely checked during and after every run of the data flow against a set of user-defined quality control rules to make sure the data flow meets the *expected behaviour* and quality criteria over time, as established upfront by the quality manager. MoniQ introduces a *monitoring description language* capable of (i) describing the semantic and the time ordering of the observational intents and capture the essence of the DI data flows to be monitored; and (ii) describing monitoring intents over the monitoring flows in terms of metrics to be extracted and controls to be ensured. The novelty of the language is that it incorporates the essence of existing data quality monitoring approaches, identifies and captures process monitoring scenarios, and, above all, provides abstractions to represent monitoring scenarios that combine data and process quality monitoring within the scope of a data flow. The study is provided with an extensive analysis of two real-world use cases used as support and validation of the proposed approach, and discusses an implementation of MoniQ providing quality managers with high-level tools to integrate the solution in a DI in an easy, technology transparent and cost efficient way in order to start to get insight out data flows by visualizing the trends of the metrics defined and the outcome of the controls declared against them.

Sommario

NELL'ultimo decennio, ricercatori, organizzazioni e finanziatori di tutto il mondo hanno rivolto la loro attenzione alla realizzazione di *Data Infrastructure (DI)*, infrastrutture digitali volte alla condivisione ed al riuso di dati e prodotti della ricerca, e necessarie a supportare le attività di una comunità di ricercatori con l'ampia gamma di servizi da essa richiesti. Le infrastrutture dati sono intese in questo lavoro come (eco)sistemi ICT in grado di offrire *componenti/servizi di processamento e persistenza dati* che possono venire combinati in *data flow* in modo tale da rendere possibile una manipolazione dati arbitrariamente complessa atta a servire le richieste degli utenti della DI, siano essi umani o macchine.

I dati risultanti dall'esecuzione dei data flow rappresentano una risorsa primaria sia per gli utenti della DI, che richiedono i risultati attesi, sia per la comunità o organizzazione che la mantiene, la cui esistenza e sostenibilità in termini di costi spesso dipende strettamente dalla usabilità ed dalla efficienza della DI stessa. Al tempo stesso, tuttavia, la gestione nel tempo dell'esecuzione di molteplici (possibilmente concorrenti) data flow apre ad una serie di problematiche ben note agli esperti che possono compromettere il funzionamento della DI, minarne l'affidabilità e non soddisfare le richieste degli stakeholder. Queste problematiche gettano radici comuni in una serie di cause, spesso tra loro correlate o combinate, quali (i) la mancanza di garanzie di sorta – in termini di qualità, stabilità dei dati, etc. – da parte delle sorgenti esterne aggregate, che tipicamente non sottostanno al controllo diretto della DI; (ii) il verificarsi di errori inattesi a qualsiasi livello di astrazione nel data flow; e (iii) la natura in continuo cambiamento dell'infrastruttura, in termini di composizione dei data flow e degli algoritmi/configurazioni in uso.

L'autonomia dei componenti di una DI, il loro (ri)uso in più data flow e l'evoluzione nel tempo dei requisiti utente rendono quello delle DI e dei loro data flow un contesto critico e soggetto alle più disparate inconsistenze. È quindi di vitale importanza fornire garanzie in termini di correttezza sul comportamento dei data flow nel tempo – garanzie quantificate in base alla qualità dei dati prodotti e alla qualità dei processi svolti – in modo tale da individuare comportamenti inesatti, prevenire e risolvere le inconsistenze o quantomeno giustificare le ragioni di tali errori, talvolta non dipendenti in toto dalla DI,

ai propri stakeholder. In aggiunta il monitoraggio può essere vitale per il funzionamento dell'infrastruttura stessa, in quanto impiegare attivamente risorse hardware e software nel processamento di dati di scarsa qualità comporta una inefficiente allocazione di queste ed uno spreco di tempo e denaro.

Tuttavia, il monitoraggio della qualità dei data flow è ulteriormente complicata dalla natura ibrida e distribuita delle DI che spesso consistono in patchwork di componenti potenzialmente sviluppati da parti distinte con distinti cicli di vita, le cui interazioni sono regolate da policy ed accordi stabiliti a livello infrastrutturale. È proprio a causa di questa eterogeneità che spesso le DI sono sprovviste di strumenti di monitoraggio in questo senso e i *quality manager* si vedono costretti ad integrare a posteriori soluzioni *ad-hoc* facendosi carico di ingenti costi di sviluppo e manutenzione.

In questa tesi, introduciamo MoniQ, una soluzione general-purpose al problema del monitoraggio della qualità dei data flow, in grado di monitorarne i componenti critici e di controllarne costantemente le metriche vitali durante e a seguito di ogni esecuzione contro un insieme di vincoli definiti dall'utente al fine di verificare che il data flow si comporti, nel tempo, nel rispetto delle specifiche dettate dai quality manager. MoniQ introduce un linguaggio descrittivo di monitoraggio in grado di (i) specificare la semantica delle metriche e con quale ordinamento temporale queste intendono essere campionate dal data flow sottostante; e (ii) descrivere gli intenti di monitoraggio in termini di metriche da estrarre, aggregazioni, composizioni e controlli da effettuare. La novità introdotta dal linguaggio consiste nel mettere a fattor comune, in un unico scenario di monitoraggio, l'essenza degli approcci esistenti per il monitoraggio sia della qualità dei processi che della qualità dei dati. Lo studio è corredato da un'analisi estensiva di due casi d'uso reali impiegati a supporto e validazione dell'approccio proposto, e descrive una possibile implementazione di MoniQ che fornisce ai quality manager strumenti di alto livello che consentano un'integrazione facile e dai costi contenuti della soluzione e che mettano in breve tempo in condizione di ispezionare i trend delle metriche estratte dai data flow e valutare l'esito dei controlli specificati su di essi.

List of publications

International Journals

1. Artini, M., Atzori, C., Bardi, A., La Bruzzo, S., Manghi, P. and Mannocci, A. (2015). The OpenAIRE Literature Broker Service for Institutional Repositories. In *D-Lib Magazine*, 21(11), 3.

2. Manghi, P., Artini, M., Atzori, C., Bardi, A., Mannocci, A., La Bruzzo, S., Candela, L., Castelli, D. and Pagano, P. (2014). The D-NET software toolkit: A framework for the realization, maintenance, and operation of aggregative infrastructures. In *Program*, 48(4), 322-354.

International Conferences/Workshops with Peer Review

1. Mannocci, A. and Manghi, P. (2016, September). DataQ: A Data Flow Quality Monitoring System for Aggregative Data Infrastructures. In *20th International Conference on Theory and Practice of Digital Libraries* (TPDL) (pp. 357-369). Springer International Publishing.

2. Mannocci A., Casarosa V., Manghi P. and Zoppi F. (2016, January). The EAGLE data aggregator: data quality monitoring. In *7th EAGLE International Conference*.

3. Mannocci, A., Casarosa, V., Manghi, P. and Zoppi, F. (2015, January) The EAGLE Europeana network of Ancient Greek and Latin Epigraphy: a technical perspective. In *11th Italian Research Conference on Digital Libraries* (IRCDL).

4. Mannocci, A., Casarosa, V., Manghi, P. and Zoppi, F. (2014, November). The Europeana Network of Ancient Greek and Latin Epigraphy Data Infrastructure. In *8th Research Conference on Metadata and Semantics Research* (MTSR) (pp. 286-300). Springer International Publishing.

5. Casarosa, V., Manghi, P., Mannocci, A., Rivero Ruiz, E. and Zoppi, F. (2014) A Conceptual Model for Inscriptions: Harmonizing Digital Epigraphy Data Sources. In *1st EAGLE International Conference on Information Technologies for Epigraphy and Digital Cultural Heritage in the Ancient World* (pp. 29-30).

Others

1. Mannocci A. (2015). Data Flow Monitoring: system requirements and design. *OpenAIRE2020*. Deliverable D8.4.

2. Mannocci A. (2014, November). Monitoring data quality in research data infrastructures. In *4th RDA plenary meeting*. Poster session.

List of Abbreviations

A

ADI	Aggregative Data Infrastructure. 47

D

DFD	Data Flow Diagram. 3
DI	Data Infrastructure. 1–5, 7–10, 12–15, 21–23, 29, 30, 34, 36–39, 41–53, 57–59, 61–63, 65, 66, 88, 89, 91, 93–98, 103, 104, 106, 107
DIS	Data Integration System. 26

H

HITL	Human-In-The-Loop. 34, 35, 43
HPC	High Processing Computing. 29

I

IMS	Information Manufacturing System. 3, 7
IQ	Information Quality. 27

O

OA	Open Access. 8, 10, 11, 20, 21, 74, 78, 81

Q

QoS	Quality of Service. 23

S

SoA	Service-oriented Architecture. 10, 11
SWfMS	Scientific Workflow Management System. 6, 24, 27–29, 35, 106, 107

V

VRE Virtual Research Environment. 2

W

WebUI Web User Interface. 7, 8, 21, 88–90, 93–95, 97, 105

WEP Workflow Execution Plan. 27, 35, 107

WfMS Workflow Management System. 6, 24, 25, 29, 31, 35, 43, 107

Contents

CHAPTER 1

Introduction

In this chapter we will introduce context, motivations, challenges, and contribution of the present thesis. Section 1.1 will introduce the notion of Data Infrastructures (DIs) as intended and adopted in this work, while Section 1.2 will describe the major problems arising when it is necessary to monitor the internal status and the data flows in such infrastructures. Finally, in Section 1.3, we describe our contributions to the research field and describe the main technical challenges we had to face.

1.1 Data infrastructures

In the last decade, a lot of attention worldwide has been brought by researchers, organizations, and funders on the realization of *digital infrastructures* or *e-infrastructures*, namely systems supporting researchers with the broad spectrum of resources they need to perform science, such as (i) hardware resources (e.g. networks, storage, computing resources, clouds, grids), (ii) system-level middleware resources (e.g. service registries, credential delegation services, certificate authorities), (iii) processing resources (e.g. software, applications, tools, services), and (iv) data resources (e.g. repositories, archives, databases). e-Infrastructures are by definition "systems of systems", whose resources may not constitute one coherent information system, but may be simply brought together by common practices and needs of the researchers of a community. Today most of science is conducted in a computation-driven or data-intensive fashion [45, 48, 107] and the realization of powerful e-Infrastructure information systems or the improvement of existing ones towards becoming more accessible information systems is of crucial importance to accelerate science and optimize its costs. Evidence of this trend are: the Cyberinfrastructure programme launched by the US National Science Foundation (NSF),

which planned[1] to develop new research environments in which advanced computational, collaborative, data acquisition and management services are made available to researchers connected through high-performance networks; the European Strategy Forum on Research Infrastructures (ESFRI), which presented the first European roadmap[2] for new, large-scale Research Infrastructures; and the European e-Infrastructure Reflection Group (e-IRG)[3], which identifies guidelines and recommendations towards the realizations of e-Infrastructures where the principles of global collaboration and shared resources are intended to encompass the sharing needs of all research activities.

Although several definitions of "Data Infrastructure" can be found in the literature, which may differ depending on the level of abstraction (e.g. services, middleware, software, hardware) and broadness of their interpretation (e.g. discipline, cross-discipline, general-purpose), these explicitly are built on, or include, the definition of e-Infrastructure. For example, in the context of scientific research and scholarly communication, the notion of (Aggregative) Data Infrastructure (ADI) [7, 57, 84] has been given as "e-Infrastructures serving scientific research communities and created to exploit the increasing and diffused online presence of research data scattered across multiple data sources, the cross-discipline nature of science and the need of researchers to gain immediate access to research all material". With a broader perspective, the European Commission issued a vision document for Global Research Data Infrastructures (GRDI)[4], which identifies a roadmap technical and organizational recommendations for designers and developers of future research DIs seen as e-Infrastructures connecting data archives, library services, and community services. In summary, there seem to be consensus on defining DIs as "e-infrastructures promoting data sharing and consumption, needed for the operation of a society/community as well as the services and facilities necessary for data economy to function"[5].

Examples of DIs tailoring scientific communication are OpenAIRE[6] [58, 94] and CORE-UK[7] [51]. Such systems deliver the aggregation of thousands data sources to enable cross-source, cross-discipline, cross-continent discovery and, whenever possible, the construction of citation indexes. Other examples from thematic-domain supporting access to different research products in Cultural Heritage are Europeana[8], the European Film Archive [4], the Heritage of the People's Europe [7], the Europeana network for Ancient Greek and Latin Epigraphy[9] [60]; while other examples supporting scientific research activities are D4Science[10] – hosting more than 50 Virtual Research Environments (VREs) to serve the biological, ecological, environmental, and statistical communities world-wide [25] – the several instances of *science gateways*[11] and one-stop shops serv-

[1] NSF Cyberinfrastructure Vision for 21st Century Discovery, `http://lib.colostate.edu/publicwiki/images/c/cf/CI_Vision_March07.pdf`

[2] ESFRI roadmap 2016, `http://www.esfri.eu/sites/default/files/20160309_ROADMAP_browsable.pdf`

[3] e-IRG "blue paper" (2010), `http://e-irg.eu/documents/10920/238805/e-irg_blue_paper_2010`

[4] Global Research Data Infrastructures (GRDI), `http://www.grdi2020.eu/repository/filescaricati/6bdc07fb-b21d-4b90-81d4-d909fdb96b87.pdf`

[5] `https://en.wikipedia.org/wiki/Data_infrastructure`

[6] OpenAIRE, `http://www.openaire.eu`

[7] CORE - The UK Open Access Aggregator, `https://core.ac.uk`

[8] Europeana, `http://www.europeana.eu`

[9] EAGLE project, `http://eagle-network.eu`

[10] D4Science, `https://www.d4science.org`

[11] Science Gateways `http://sciencegateways.org/about/science-gateway-basics`

ing an entire community, such as Hola Cloud[12], nanoHUB[13] [50], HUBzero[14] [67], the Protein Data Bank (PDB)[15] [17], Galaxy[16] [40] and many others.

In this thesis, we look at DIs from the perspective of their data *processing components* – e.g. web services, desktop tools, libraries – and how such components are organized into *data flows* so as to manipulate and produce data stored in and provided by *data components* – e.g. databases, document stores, external data sources, data streams – thereby excluding from the scope lower layers of the DIs stack, mainly relative to hardware and deployment issues. In the following, we shall refer to the following general definitions:

Definition 1.1. *Data Infrastructures (DIs)* are here intended as ICT (eco)systems constituted by *data* and *processing components* which can be combined into *data flows* so as to enable arbitrarily complex data manipulation actions serving the *consumption needs* of DI customers, be them humans or machines. □

Definition 1.2. *Processing components* logically represent software agents whose execution, fired by a human or machine, is intended to process given input data. □

Since we are dealing with digital science, a processing component does not represent the action of a human in a laboratory. Examples of processing components can be a web service, a local procedure, a desktop tool or a web application to be manually activated, etc.

Definition 1.3. *Data components* logically represent "data stores" used to handle "collections of objects" and living in the DI. They are used as sources or targets of processing components. □

Examples of data components can be relational databases, indexes, streaming sources, file systems, document stores, object stores, HBase stores, etc.

Definition 1.4. A *data flow* is the specification of "how" a given DI data processing goal is achieved in terms of an ordered sequence of (possibly parallel) processing components applied over data components of the DI. A data flow is completely (or partly) executed when all (or part) of its processing components are executed. □

For the sake of clarity, by *data flow* we intend a more detailed, data-perspective-oriented view of a *workflow*, classically intended as a sequence of *activities* ordered according to casual and data dependencies. Inspired by the paradigms introduced by Data Flow Diagrams (DFDs) and Information Manufacturing Systems (IMSs) [6, 95], a data flow can be seen as the specification of a workflow that includes also the data components involved. As will be noticed in the following chapters, this terminology will also help us to make a clear-cut distinction between *workflow monitoring* and *data flow monitoring* as intended in this work.

It has to be noted that this definition makes no assumption on how components should be implemented and if data flows are *automated*, i.e. based on orchestration mechanisms, *manual*, i.e. consisting a sequence of manual steps, or *hybrid*, i.e. partly manual and

[12] Hola Cloud, `http://www.holacloud.eu`
[13] nanoHUB, `https://nanohub.org`
[14] HUBzero, `http://hubzero.org`
[15] Protein Data Bank, `http://www.rcsb.org`
[16] Galaxy, `http://galaxyproject.org`

partly automated. All abstractions above are logical, as are not intended to capture the level of component gluing, automation, common middleware, common policies, etc. as well as the data models, back-end peculiarities, or exchange formats. In summary, we are interested to model the expected behaviour of DIs in terms of the critical data flows which are daily supporting its users. For that we aim at representing the data and processing components of a DI, expressing their time ordering and manipulation business logic in terms of data flows, as conceived and realized by the collaboration of multiple DI actors, with possibly overlapping roles, such as *administrators*, *designers and programmers*, *policy managers* and *quality managers*; being the latter the main addressee of this work. The high-level view supported by the model is generic enough to express the behaviour DIs of research-oriented and commerce-oriented e-Infrastructures, independently of their technological maturity, technical integration and cohesion, and development platforms.

1.2 Monitoring data infrastructures: the challenge

Data resulting from the execution of data flows, as defined in Definition 1.4, represent an important asset both for the DI users, typically craving for the information they need [6, 39, 120], and for the organization (or community) operating the DI, whose existence and cost sustainability depends on the adoption and usefulness of the DI. On the other hand, when operating several data processing workflows over time, several issues, well-known to practitioners, may arise and compromise the behaviour of the DI, and therefore undermine its reliability and generate user dissatisfaction [36, 38, 88]. Data components and relative collections may disappear or be altered because of reasons unpredictable by the DI *quality manager(s)*, for example: (i) an entire collection or a portion of it could be lost because of an hardware failure (e.g. unexpected hard drive or network failures) or because it has been moved somewhere else by another data flow or deleted by accident; (ii) data sources, whose control is often out of the scope of the DI, do not always provide guarantees (e.g. SLAs) about the expected quality level of the exported data or the stability of their collections; (iii) when produced in high volumes by hardware instrumentation, data are subject to technological issues (e.g. geo-location failure, quality degradation in the due to some transient error, hardware failures, calibration drifts and so on).

Similarly, the computations underlying processing components may not behave as expected, for example: (i) subtle programming imprecisions (e.g. overlooking exceptions in the data to be processed) may introduce errors in the resulting data which are often invisible to the naked eye within a single computation and may degenerate over time; (ii) minor inconsistencies in the input data (e.g. conformance of the exchange format) may prevent part of the data to be processed and therefore compromise the quality of the results.

Finally, data flows combining data and processing components may suffer from combinations of such inconsistencies, which may manifest in an individual execution session of a data flow, or across several sessions over time. For example, a data flow may consist of multiple parallel processing components contributing to the population of the same data component collection, where data provenance [23, 29, 99] is not tracked; the expected behaviour of the data flow is that the total size of this collection should

be provided with given expected percentages by the individual processing components. Monitoring the behaviour of individual processing components would not suffice to validate this constraint, as well as monitoring the target data component collection. Processing and data components should be individually monitored, but controls over their combined behaviour should be applied within a scope which includes all of them. In a more elaborated scenario, DI quality managers may be requested to verify that such percentages are respected with a tolerance of 1 disruption out of 10 executions of the data flow. In this case, controls should therefore be applied in a temporal scheme, taking into account past observations and measurements over the data flow components.

The autonomy of DI components, their use across several data flows, the evolution of end-user requirements over time, make the one of DI data flows a critical environment, subject to the most subtle inconsistencies. Accordingly, DI users demand guarantees, while quality managers are called to provide them, on the "correctness" of the DI data flows behaviour over time, to be somehow quantified in terms of "data quality" and in terms of "processing quality" [6, 88, 102, 105]. Monitoring data flows is therefore a key activity of paramount importance to ensure the up-taking and long term existence of a DI. Indeed, monitoring can detect or anticipate misbehaviours of the DI, in order to prevent and adjust the errors or at least "formally" justify to the stakeholders the underlying reasons, possibly not due to the DI, of such errors. Not only, monitoring can also be vital for DI operation, as having hardware and software resources actively employed in processing low quality data can yield inefficient resource allocation and waste of time [66, 89, 91].

As we have seen above, the quality of the data to be processed and produced and the quality of the processes elaborating data are closely related to the success and effectiveness of the DI data flows. Data flows demand a specific form of monitoring, which we may refer to as *Data Flow Quality Monitoring*:

Definition 1.5. *Data Flow Quality Monitoring* is a practice in which, given a DI data flow, its critical data components, as well as critical processing components involved in their elaboration, are routinely checked during and after every run of the data flow against a set of user-defined quality control rules to make sure the data flow meets the *expected behaviour* and quality criteria over time, as established upfront by the quality manager. □

DI Data Flow Quality Monitoring is complicated by the "hybrid" nature of such infrastructures, which typically consist of a *patchwork* of individual components ("system of systems") possibly developed by distinct stakeholders with possibly distinct lifecycles, evolving over time, whose interactions are regulated mainly by shared policies agreed at infrastructural level. Due to such heterogeneity, generally DIs are not equipped with built-in monitoring systems and DI quality manager must therefore introduce such capabilities as *ex-post* software integration. The literature on system monitoring is rather rich and offers a number of solutions in the direction of monitoring the "quality" of hardware resources, data, and workflows. Typically, the relative monitoring challenges are dealt with separately, as uncorrelated problems, or span across hardware-applications (workflows monitoring) and hardware-data (back-end optimization and monitoring), and are often specific to an application context or given technology.

Regarding hardware monitoring, several off-the-shelf, both commercial and open source, solutions are available at "system administration level" (e.g. Nagios, Icinga,

Ganglia), but they can do little when the focus is on the application level or, in particular, on data content and its quality. Application monitoring frameworks such as Riemann[17], Prometheus[18], DataDog[19], New Relic[20] and the ELK[21] or TICK[22] stacks work great for exporting metrics from the application level, but again they are mainly devised for platform operations, availability checks and performance monitoring (e.g. response time, counters, volume of data transferred, etc.), while fail to capture typical problems of data quality, extract meaningful metrics from data processed by the infrastructure (e.g. completeness of a dataset record) and put them in a data-flow-aware perspective, as metrics are essentially "flat" for these tools, since they are conceived for monitoring the application as an end-system.

As for data quality, it is generally agreed that quality falls under the *"fitness for purpose"* principle [11, 20, 93, 96, 102, 104, 105, 114, 120]. In fact, determining whether data has quality or not mainly depends on the application they are intended for or the algorithms they are supposed to be fed to and, in general, there is no *universal* rule-of-thumb to be applied to answer the question. Several tools, either commercial or open source, have been proposed in this ambit, e.g. Attacama DQ Analyzer and DQ Center[23], Talend Open Studio for Data Quality[24], Data Cleaner[25], Data Monitoring Tool[26], Informatica Data Quality[27], Sourceforge's DataQuality tool[28] and Experian Pandora[29]. Despite these tools are valuable for getting insight about data, as well as their patterns and flaws, in common back-ends (mostly E–R databases, occasionally noSQL stores), they seldom take into account in its entirety the potentially complex data flows responsible of populating the stores in exam, hence failing to provide a thorough vision of the whole situation.

Regarding workflow monitoring, some Workflow Management System (WfMS) – and Scientific Workflow Management System (SWfMS) too – offer advanced tools for tracking the status and performances of past and present execution of workflows (e.g. D-Net [57], Airflow[30], Taverna [78], Kepler [3], Chiron [77], Askalon [34]), Trident [8], but they are in general not concerned with monitoring how workflows and relative processing components may affect or be affected by the quality of the data [53], despite a few extensions enabling "domain data analysis" in scientific workflows can be found in the literature [66]. For example, while it is possible to know exactly how much time took to execute activities of a workflow (in all its executions), it is in general not possible to log and analyze any information extracted or derived from data processed by workflow activities (e.g. compliance to a certain standard or data format).

[17] Riemann.io, `http://riemann.io`

[18] Prometheus.io, `http://prometheus.io`

[19] DataDog, `https://www.datadoghq.com`

[20] New Relic monitoring `https://newrelic.com/application-monitoring`

[21] The ELK (elasticsearch, Logstash, Kibana) stack, `https://www.elastic.co`

[22] The TICK (Telegraf, Influx, Chronograf, Kapacitor) stack, `https://www.influxdata.com/get-started/what-is-the-tick-stack`

[23] Ataccama, `https://www.ataccama.com/products/dq-analyzer`

[24] Talend Open Studio for Data Quality, `https://www.talend.com/download/talend-open-studio#t2`

[25] Data Cleaner, `https://datacleaner.org`

[26] Data monitoring tool, `http://www.uniserv.com/en/company/blog/detail/article/data-monitoring-tool`

[27] Informatica Data Quality, `https://www.informatica.com/products/data-quality/informatica-data-quality.html`

[28] Arrahtec "Open Source Data Quality and Profiling", `https://sourceforge.net/projects/dataquality`

[29] Experian Pandora, `https://www.edq.com/uk/solutions/experian-pandora`

[30] AirFlow, `http://nerds.airbnb.com/airflow`

In summary, Data Flow Quality Monitoring demands tools capable of supervising processing and data components and express constraints that regard their co-existence and combination with other components within the scope of one data flow, regarding its individual execution sessions and over time. To date, such tools are not available off-the-shelf. DI quality managers are therefore bound to use combinations of existing tools – with non trivial integration efforts – or to develop and integrate their own *ad-hoc* solutions, at high cost of realization and maintenance. Indeed, it is complicated and expensive to adopt and maintain multiple solutions for different components and combine them to deliver uniform Data Flow Quality Monitoring tools.

1.3 Thesis contribution

In this thesis we addressed the problem of Data Flow Quality Monitoring in DIs described above. Its main contributions are the following:

1. the definition of a *monitoring description language* capable of:

   - describing monitoring flows: DI quality managers can express the semantic and the time ordering of their observational intents and capture the essence of the DI data flows to be monitored, by means of *monitoring flow description* primitives;
   - describing monitoring intents over the monitoring flows: DI quality managers can specify the metrics and controls required to perform the monitoring using *monitoring intent description* primitives.

   The novelty of the language is that it incorporates the essence of existing data quality monitoring approaches, identifies and captures process monitoring scenarios, and, above all, provides abstractions to represent monitoring scenarios that combine data and process quality monitoring within the scope of a data flow.

2. the implementation of a general-purpose Data Flow Quality Monitoring system, called MoniQ (pronounced "moh-NEEK"), whose functionalities are based on the concepts introduced by the monitoring description language. MoniQ features the following desirable properties:

   - *Data flow sensitive monitoring*: monitors the defined data flow entities of data components and processing components and persists the history of monitoring events of different data flow executions throughout time;
   - *Cross-platform compatibility*: adapts to different DI technologies;
   - *Cost-effective integration*: hides the complexity of monitoring and minimizes the amount of work required to integrate the framework on top of existing DIs.

MoniQ offers DI quality managers off-the-shelf tools for data flow quality monitoring. The DI quality manager is provided with a Web User Interface (WebUI) in which he can formally describe the monitoring flows to be monitored over the infrastructure data flows as a sequence of basic building *blocks* inspired to IMSs, introduced in [6, 95]. Once the monitoring flows have been defined, the WebUI allows quality managers to assign *sensors* to the relative building blocks (or sets of them); a sensor is the mean to

introduce a set of *metrics*, each corresponding to a number of observations collected over time, of pertinence for a block, be it relative to data or processing components. Finally, the WebUI allows quality managers to define which *controls* should be enforced over the metrics declared (e.g. rates, thresholds, upper bounds, moving averages, etc.), visualize and aggregate trends of the monitored features, analyse comprehensive *reports* and receive *notifications* and *alerts* about potential troubles happening in the operation of the infrastructure.

In order to integrate MoniQ in the DI data flow, each sensor/metrics requires a corresponding *sensor hook* within the DI, intended as a software counterpart capable of extracting observations for the identified metrics and sending them to MoniQ. Given the specification of the monitoring flow and the relative metrics, sensors and controls, MoniQ lets DI quality managers focus on the business logic required to compute the metrics observations (i.e. sensor hook) and relieve them of the hassle about implementing a persistence and the control engine for assessing the observations produced. Currently the system supports two methodologies: (i) a set of predefined Java classes for a facilitated and more compact implementation of sensors, which hide the complexity required to interact with MoniQ and whose interfaces must be implemented with a reference to the relevant sensor (sensor identifier) and the metrics business logic; (ii) a REST API, which has to be called by the DI components with the required sensor identifiers and the metrics values.

MoniQ is currently used in the production system of the OpenAIRE infrastructure to ensure the expected behaviour and quality of its data flows over time since 2015. The OpenAIRE [58, 59] Data Infrastructure is funded by the European Commission to become the point of reference for Open Access (OA) and Open Science in Europe. The initiative fosters free access, share and reuse research products in Europe and beyond and provides an European data e-infrastructure linking people, ideas, projects, organizations and funding streams. The system benefits from the flexibility of MoniQ, which allows for the introduction of new data flows, new monitoring scenarios, and minimizes the amount of work required to integrate OpenAIRE services with the framework.

1.4 Outline of the thesis

The remainder of this thesis is organized as follows:

Chapter 2 introduces the reader to fully fledged monitoring problematics by presenting two real-world use cases, namely the OpenAIRE and the CORE Data Infrastructures, and describes thoroughly their critical data flows and monitoring concerns. The two proposed use cases are taken into account in this thesis as a validation instrument of the main contributions.

Chapter 3 provides the state of the art and reviews the relevant literature in the research fields of (i) *workflow quality and monitoring*, (ii) *data quality and monitoring*, and (iii) *data and workflow quality*. Finally, it outlines a taxonomy characterising monitoring systems, be them for data or workflows, according to a number of desirable properties.

Chapter 4 analyzes the requirements for the realization of a data flow quality monitoring system and presents in all its details MoniQ, its architecture and the monitoring

description language capable of modeling monitoring flows and monitoring intents over DI data flows.

Chapter 5 provides an evaluation of the data flow quality monitoring description language proposed in Chapter 4 against the two use cases presented in Chapter 2.

Chapter 6 discusses the current realization of MoniQ and its soundness and completeness in relation to the reference architecture drawn in Chapter 4, and finally introduces MoniQ-java-di, a thin-layer of Java classes developed in order to facilitate the integration of MoniQ into Java-based DIs.

Chapter 7 finally provides a summary of the thesis and its main contributions, analyze its impact and limitations, and provides an overview of possible future work, extensions and viable research directions.

CHAPTER 2

Real-world Use Cases

In order to introduce the reader to fully-fledged monitoring problematics in Data Infrastructures (DIs), in this chapter we describe two real-world use cases, namely the OpenAIRE infrastructure, described in Section 2.1, and the CORE infrastructure, described in Section 2.2. For both initiatives, we will provide the context and outline the data flows that can take benefit from data flow quality monitoring. Both the two initiatives are relative to the world of scholarly communication and bibliographic metadata aggregation; however, the monitoring concerns here outlined are totally generic and applicable at broad spectrum. Despite the similar mission of the proposed use cases, their implementations are profoundly different from both an architectural and a technological point of view. Nonetheless, they share similarities in data flow quality monitoring concerns and this helped us in designing a monitoring systems that can be adapted to different needs.

2.1 The OpenAIRE data infrastructure

The OpenAIRE[1] [58, 59] Data Infrastructure is funded by the European Commission to become the point of reference for Open Access (OA) and Open Science in Europe. The initiative fosters free flow, access, share and reuse research products in Europe and beyond and provides an European data e-infrastructure linking people, ideas, projects, organizations and funding streams.

From a technological perspective, OpenAIRE operates an open and collaborative Service-oriented Architecture (SoA) enabling:

- the realization of a pan-European network for the definition, promotion and implementation of shared interoperability guidelines and best practices for managing

[1] The OpenAIRE project, `http://www.openaire.eu`

sharing, reusing and preserving research outcomes of different typologies;

- the promotion of Open Access and Open Science and practices at all stages of the research lifecycle and across research communities taking part to different application domains and geographic areas;
- the generation of statistics measuring the impact of Open Access and Open Science as well as their return on investment (RoI) across Europe and individual countries joining the initiative;
- the creation of a centralized entry point for discovering people, publications, projects, and datasets produced as scientific output within H2020 (as well as FP7 and others funding schemes) and browsing their interconnections.

The SoA provided by OpenAIRE is realized thanks to the tools and framework provided by the D-NET software toolkit [57]. The main programming language employed is Java, but a few other specific parts have been developed using other languages too such as Perl, Python and Groovy[2]. The infrastructure is operating in a regimen of continuous data integration (i.e. 24/7/365) and feeds a public graph-like information space capable of serving the aforementioned use cases.

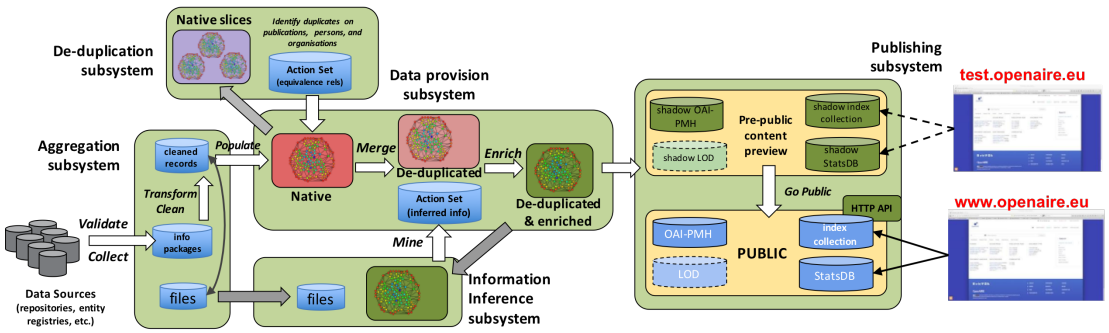


Figure 2.1: *The OpenAIRE functional architecture*

The OpenAIRE infrastructure features a number of subsystems, reported in Figure 2.1, dedicated to four main activities:

- **Aggregation subsystem**: the subsystem takes care of the collection of information packages about different entities declared in the OpenAIRE data model and publication full-texts (e.g. PDFs, XMLs, HTMLs) from data sources. Based on the typology of such information packages (e.g. Dublin Core[3] metadata records, DataCite metadata records, CERIF-XML metadata records, proprietary formats), the system transforms them into "cleaned" metadata records with uniform structure and semantics, matching the specification of the OpenAIRE data model.
- **Deduplication subsystem**: given as input the native information space graph as yielded by the data provision subsystem (see below), the system identifies duplicates among the objects of the same entity type. For each entity among publications, organizations and persons, the system generates a set (called *actionset*)

[2]The Groovy programming language, `http://www.groovy-lang.org`
[3]Dublin Core Metadata Initiative, `http://dublincore.org`

of similarity relationships between pairs of objects identified as duplicates, which will be used by the data provisioning subsystem to generate a disambiguated information space.

- **Information inference subsystem**: given as input the last public information space graph (deduplicated and enriched by provision in the last round) and the publications full-texts, the system applies a number of mining algorithms (i.e. "modules") over such data corpora. For each mining module the system produces a set of inferred knowledge (another actionset), which will be used by the data provision subsystem in order to enrich the information space graph. The enrichment regards, for example, (i) new entities (e.g. publications), (ii) new attributes for existing entities (e.g. metadata attributes for publications), (iii) new relationships among entities (e.g. links from publications to projects, publication similarity, citations among publications, links among publications and datasets, affiliations between publications and organizations).

- **Data provision subsystem**: given as input the cleaned metadata records as yielded by the aggregation subsystem, the similarity relationships as (lastly) yielded by the deduplication subsystem, and the inferred knowledge as (lastly) yielded by the information inference subsystem, the data provision subsystem: (i) populates an initial bare-aggregation information space graph from cleaned metadata records (i.e. native information space graph), (ii) enriches the graph with similarity relationships and runs an object merging algorithm to remove duplicates, (iii) enriches the graph with inferred information. The information graph ultimately produced (deduplicated and enriched) is the next candidate to become publicly accessible.

- **Publishing subsystem**: given as input the deduplicated and enriched information graph as produced by the data provision subsystem, the data publishing subsystem materialize the graph over four different backends serving different use-cases: (i) full-text index supporting portal queries, (ii) a document store supporting and OAI-PMH publisher, (iii) a E-R database supporting statistics for funders, (iv) and triple-store for Linked Open Data (LOD) export (this effort is currently under development within the OpenAIRE2020 project). For the sake of prudence, the four projections are written into "private" (i.e. shadow) collections, not yet publicly accessible, into the four backends. Once the four materialization processes have ended and their products have been checked, a manual switch brings the newly generated information space online.

Talking about mere figures the OpenAIRE infrastructure features, a per-entity breakdown can be found in Table 2.1.

Given the complexity of the DI as well as the amount of information routed, certifying the quality of "data products" and how much reliable is the information provided to the end-users is a non-trivial, yet vital, task for providing trustful and valuable search and statistical services.

However, data quality in OpenAIRE is dependent from a plethora of variables such as the variety and variability of data source, the refinement of algorithms currently in use for curation and mining, the cross-influence between the presence/absence of entire sets of information and the chosen mining/deduplications algorithms, the presence of network and I/O errors, etc.

Table 2.1: *OpenAIRE in figures (as in August 2016)*

	Total amount
data sources	6,081
publications	16,923,747
full-text articles	4,099,925
persons	13,727,371
organizations	62,798
projects	652,384
datasets	23,830

For all these reasons, OpenAIRE, being indeed a DI featuring such serious monitoring challenges, has been chosen and analyzed in order to extract valuable use cases for our study; a representative selection of them, organized per subsystem, is exposed in the following sections.

2.1.1 Monitoring the aggregation data flows

In the OpenAIRE infrastructure, the aggregation subsystem features different data flows dealing with the collection of native records from data sources, their transformation and (optional) cleaning. The different data flows deal with different types of data sources and different types of collected information packages:

- **Literature data sources**:
  - **Institutional or thematic repositories**: information systems where scientists upload the bibliographic metadata and PDFs of their articles, because of obligations with their organization or because of research community (good) practices (e.g. ArXiv, EuropePMC);
  - **Third-party literature aggregator services**: information systems that, like OpenAIRE, collect descriptive metadata about publications from multiple sources in order to enable cross-data source discovery of given research products. Such aggregators tend to be driven by research community needs or to target the larger audience of researchers across several disciplines. Some examples are BASE[4] for scientific publications and DOAJ[5] for Open Access journals publications;
  - **Open Access Publishers**: information system of open access publishers or relative journals, which offer bibliographic metadata and PDFs of their published articles;
- **Scientific data sources**:
  - **Data archives**: information systems where scientists deposit descriptive metadata and files about their research data (also known as scientific data, datasets, etc.); data archives are in some cases supported by research and academic organizations and in some cases supported by research communities and/or associations of publishers;

[4]BASE, `https://www.base-search.net`
[5]DOAJ, `https://doaj.org`

  - **Third-party scientific data aggregator services**: similar to the literature aggregator services, but dealing with scientific data mainly. An example is DataCite[6] for all research data with DOIs as persistent identifiers.

- **Entity registries**: information systems created with the intent of maintaining authoritative registries of given entities in the scholarly communication, such as OpenDOAR[7] for the institutional repositories or re3data.org[8] for the data repositories;

- **CRIS systems**: Current Research Information Systems (CRIS) are adopted by research and academic organizations to keep track of their research administration records and relative results; examples of CRIS content are articles or datasets funded by projects, their principal investigators, facilities acquired thanks to funding, etc.

The data flows collecting from such sources are probably one of the most sensitive ones in the whole infrastructure as they superintend the first (and foremost) entry point for "alien" data into OpenAIRE. In fact, data can possibly carry along imperfections or suboptimal and unwanted data features that can invalidate the assumptions and the premises for a good operation of the DI. As we are interested in controlling whether certain characteristics of the data and the processes hold or not, a thorough monitoring of these data flows is vital. In the following we will describe monitoring concerns divided by data source typologies of interest, namely, at the time of writing, *literature data sources* and *entity registries*.

Literature data sources

The aggregation data flow for literature data sources can be described in a nutshell as in Figure 2.2. The data flow harvests literature metadata from a publicly accessible OAI-PMH[9] endpoint, transforms and cleans the metadata records compatibly to OpenAIRE specifications and stores the results produced in a document store dedicated for each data source. In parallel, another process takes care of downloading PDF of the research products, when available. Such data flows is executed in an independent instance for each one of the literature data sources registered in OpenAIRE.

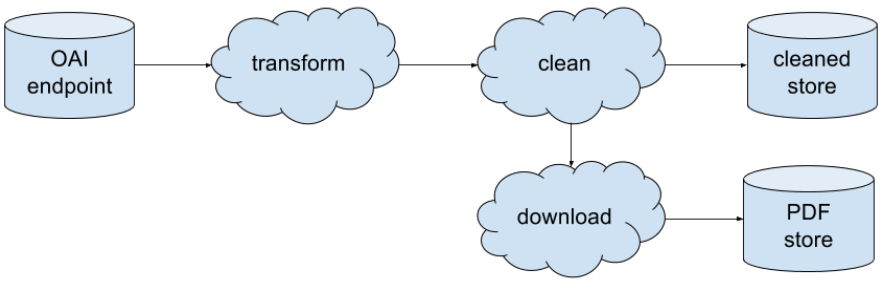


Figure 2.2: *OpenAIRE aggregation data flow for literature data sources*

[6]DataCite, `https://www.datacite.org`
[7]OpenDOAR, `http://opendoar.org`
[8]Re3data, `http://www.re3data.org`
[9]OAI-PMH, `https://www.openarchives.org/pmh`

This kind of data sources (e.g. institutional repositories, literature aggregators, digital libraries, etc.) provides OpenAIRE with metadata records about research literature (e.g. publications, technical reports, deliverables, etc.). As research activities go further and new literature got published, research products are expected to be registered into the data sources; thus, the amount of harvestable data is expected to be increasing over time; monotonically increasing to be precise. Any alteration in such kind of trend might be caused by an overlooked situation and the event should be promptly detected and flagged as a warning. This gives us a first metrics of interest accounting of the total number of metadata records available over time at a generic data source.

Moving forward, literature metadata records comply to the Dublin Core standard. Records must be checked for quality according to metrics of interest as they firstly reach the OpenAIRE DI. Such analysis can be interesting both for both new records, as fresh data entry error could have been made, and older ones as corrections and/or accidental modifications could have been performed. This outlines, for example, the interest in monitoring the *completeness* of records, e.g. according to a pool of expected initialized fields such as title, author, keywords, description, year of publication, and *compliance* w.r.t. OpenAIRE guidelines[10] and analyse how these two metrics evolve over time as a valuable indicator of the "health" of a generic data source.

Finally, the aggregation data flow also takes care of cleaning incoming literature metadata records. The cleaning process takes care of aligning the content of certain known fields (vocabulary-controlled fields) to agreed vocabularies defined upfront in OpenAIRE. In this sense, another interesting metrics to be monitored would indicate how well the cleaning process performs; the metrics could track for example the number of changes (i.e. corrected field's content) performed by the process on average per-record. Another valuable metrics could track the number of fields that the cleaning process did not achieve to align (e.g. because they are not known).

Entity registries

The data flow aggregating from entity registry, represented in Figure 2.3, collects the information packages provided by the entity registry at hand, transforms and cleans the records and stores them into OpenAIRE database.

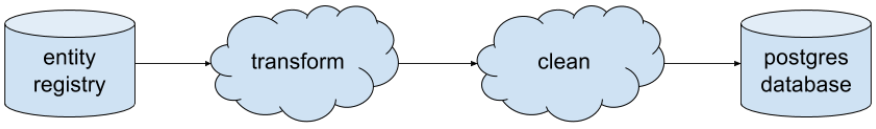


Figure 2.3: *OpenAIRE aggregation data flow for entity registries*

Each entity registry contributes to OpenAIRE with a specific kind of information package providing a particular set of entities of the OpenAIRE data model. For example, an instance of this data flow periodically aggregates records about FP7 and H2020 projects from Corda[11]. For each record collected, a transformation is applied and participating organizations are extracted. The information about projects and participating

[10] OpenAIRE guidelines, `http://guidelines.openaire.eu`

[11] The CORDA (Common Research DAta Warehouse) database contains data on applicants/proposals and signed grants/beneficiaries with regards to the European Framework Programme for Research.

organization is then stored in a relational database. It is important that subsequent executions of this data flow show specific trends at the end of the data flow. In particular, the number of projects and the number of organizations is expected be monotonic increasing ensuring that any record has been removed by accident at the source; a thing that in a few occasions happened and invalidated the reasoning produced by the inference modules.

It is also important to keep under control the number of $misdatedproject$, meaning that its duration falls within the funding stream time extension. For example, an FP7 project record might advertise a duration from 2005 to 2008 which clearly does not comply with FP7, which covers from 2007 to 2015. The number of misdated projects should be monotonically decreasing over time.

Moving further, it would be interesting to monitor the outcome of the cleaning process. The cleaning process tries to fix the content of some fields in project records according to a set of given controlled vocabularies. If the content of a field at hand cannot be aligned to one of the admitted values by the vocabulary controlling the field, then the cleaning process simply leaves the field content untouched and the record passes through. In this case, it would be valuable to monitor the average degree of $compliance$ (to the defined controlled vocabularies) of project records leaving the cleaning process.

Finally, it would be useful to assess the $completeness$ of the two tables about projects and organizations stored in PostgreSQL. An average completeness lower than 0.7 would mean scarcely initialized data in input.

2.1.2 Monitoring the deduplication data flow

The deduplication subsystem takes care of the delicate processes of finding duplicated entities in the OpenAIRE information space. Hence, the whole deduplication data flow,

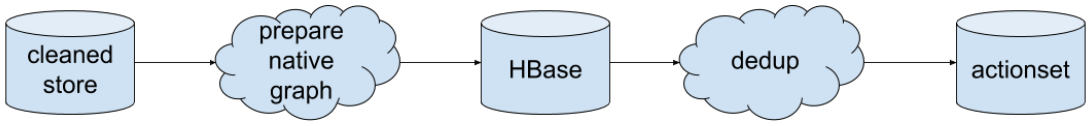


Figure 2.4: *OpenAIRE deduplication data flow*

represented in Figure 2.4, takes care of a delicate task to carry out as false positives (and false negatives) produced by the deduplication discovery algorithm may alter over time the quality (in terms of counts and in terms of hidden records) of the similarity relationships produced in output and the resulting data published to the general public and to the European commission. In particular, it is important to track the deduplication factor of several entities of interest such as *publications*, *organizations* and *authors*. The *deduplication factor* takes into account how many entities (either publications, organizations or authors) are merged into one representative by the deduplication process. More precisely, if the deduplication process claims that e.g. five publications are considered to be akin, it clusters the five publication records and generates one representative with merged metadata.

During the development and fine tuning of the deduplication algorithm for publications, we identified that, on average, 2.4 publications get merged together [5], as can be seen from Figure 2.5. This could be used as a reference threshold: every time the

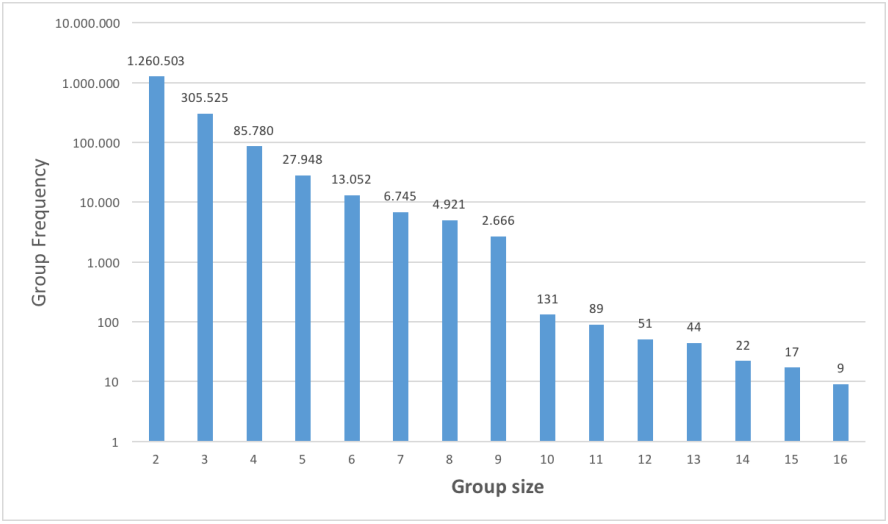


Figure 2.5: *Frequency of group sizes for deduplication of publications*

deduplication process yields a average deduplication factor greater than or too distant (either above or below) from 2.4, a warning should be flagged and the post-deduplication data should be double checked as there could be issues (e.g. new data or missing data invalidating deduplication reasoning, problems in the algorithm or in its configuration, etc.).

In a similar way, on average 2.14 organizations get merged by the deduplication process, as can be seen in Figure 2.6. This value can be used as well as a threshold

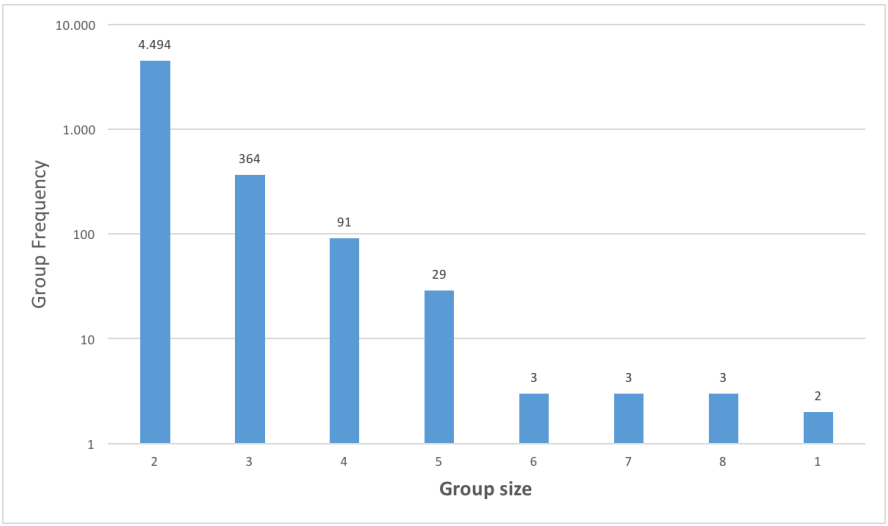


Figure 2.6: *Frequency of group sizes for deduplication of organizations*

for assessing the behaviour of the deduplication process for organizations and receive warnings whenever something performs unexpectedly.

Finally, despite the deduplication of authors is a current work-in-progress, some metrics (e.g. deduplication factor for persons) could be tracked and used as feedback for our developers as important indicators about how the deduplication process is performing.

Although there is no need to enforce controls over such metrics at this stage, this example shows how even just plain monitoring can still be a valuable aid and drive fine tuning of the configuration of the algorithm, and/or improving its implementation.

2.1.3 Monitoring the inference data flow

The inference data flow, reported in Figure 2.7, processes the PDFs collected by the aggregation subsystem in the attempt of extracting full-texts of publications from them. Once the extraction process has terminated, the extracted full-texts and the lastly available deduplicated graph are taken as input for inference algorithms (more details about the inference subsystem can be found in [52]) that ultimately produces inferred relationships in an actionset (i.e. a set of "patches" to be applied in order to enrich the information space graph).

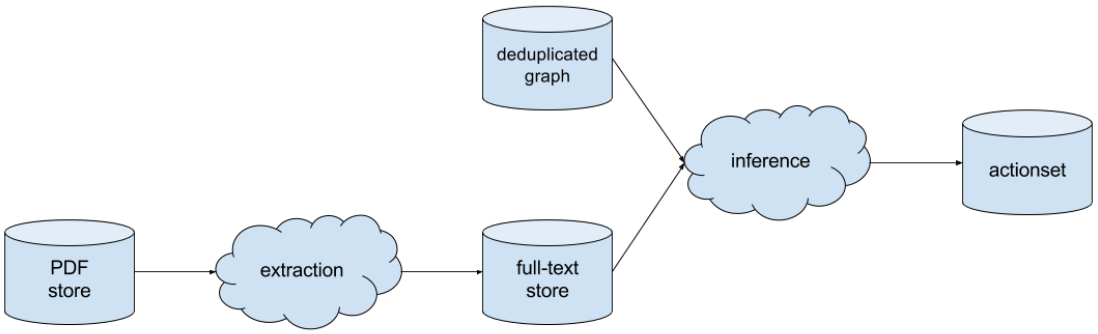


Figure 2.7: *OpenAIRE inference data flow*

It is interesting to ensure that subsequent productions of actionsets hold certain properties. For example, the number of produced relationships of each kind (e.g. links from publications to projects, publication similarity, citations among publications, links among publications and datasets, affiliations between publications and organizations) should be monotonic increasing over time and the percentage variation should not exceed a certain threshold given the increase of the amount of data in input from the last round. A potential issue should be flagged otherwise. Moreover, since mining algorithms can be enhanced or exchanged over time as well as their configurations, it is interesting to track their yielded production and verify that no drift is introduced into the system by accident.

Moving forward, another interesting monitoring concern regards PDF and full-text mining process. In its earlier years, in a few occasions, the OpenAIRE infrastructure experienced some issues because of malformed collected PDFs, hence an important aspect to monitor could regard the ratio between malformed PDFs and well-formed ones which is desired to be decreasing over time. Finally, given a well-formed PDF, it is not granted that the relative full-text can be extracted successfully; for example, PDFs regarding old publications are usually scanned page by page, hence no (or little) text can be actually extracted. For this reason it is interesting to verify whether the number of successfully extracted full-texts covers more than a significant percentage (say 90%) of the number of well-formed PDFs.

2.1.4 Monitoring the publishing data flow

The *publishing data flow*, represented in Figure 2.8, materializes the information graph (which is stored in HBase[12] as yielded by mining and deduplication subsystems) into: (i) a full-text index, implemented with Apache Solr[13] to support search and browse queries from the OpenAIRE Web portal, (ii) a PostgreSQL[14] and a dedicated Redis[15] key-value cache for statistics, (iii) a NoSQL document storage based on MongoDB[16] in order to support OAI-PMH export of the aggregated records, and finally (iv) a triple store realized with OpenLink Virtuoso[17] in order to expose OpenAIRE's information space as LOD via a SPARQL endpoint.

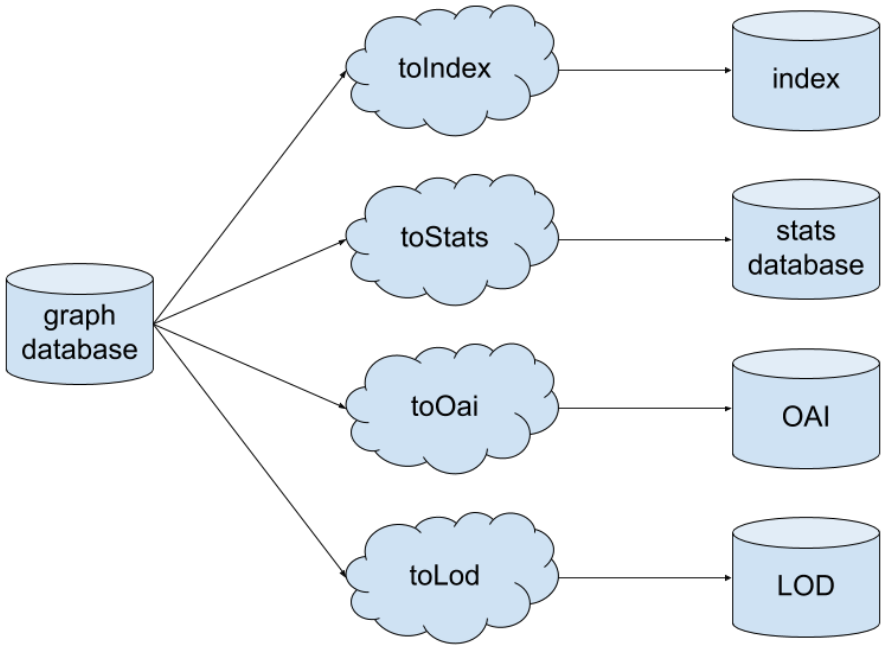


Figure 2.8: *OpenAIRE publishing data flow*

Every time the publishing data flow executes, four new information space projections are generated and placed in a "pre-public status" before being accessible from the general public. The switch from pre-public to public, meaning that the currently accessible information space projections and statistics will be dismissed and the new versions will take their place, is still manual for safety reasons.

Whenever the pre-public information space projections (pre-public information space (IS) for short) are created, it is interesting to verify some constrains in order to evaluate whether the switch to public can be performed or something has gone wrong during the data flow execution. Some conditions to be ensured are the following:

- **Snapshot checks**: right after the generation of a new pre-public IS, check whether some properties in the four backends follow user-defined constraint or not.

[12] Apache HBase, `https://hbase.apache.org`
[13] Apache Solr, `http://lucene.apache.org/solr`
[14] PostgreSQL database, `http://www.postgresql.org`
[15] Redis, `http://redis.io`
[16] MongoDB, `https://www.mongodb.org`
[17] OpenLink Virtuoso, `http://virtuoso.openlinksw.com`

  - **Inter-backend consistency checks**: check whether certain relations hold among properties extracted from different backends; for example, check if different backends are aligned w.r.t their content. This means assessing (and keep assessing over time) whether a certain property extracted from a backend matches the same property extracted in the symmetrical way from another backend. For example, check whether the total number of "publications funded by Horizon2020 projects" indexed by Solr matches the same number of records advertised via the corresponding OAI-PMH set and statistics.
  - **Intra-backend quality checks**: check whether the pre-public IS in a certain backend in a given moment satisfies a certain user-defined condition. For example, evaluating whether the "completeness" of records indexed by Solr, evaluated as the ratio between number of mandatory attributes and number of not empty mandatory attributes, is above a certain threshold or not.

- **History checks**: check whether the last *n-steps* of pre-public IS generations (or within a given time interval) of the history of given OpenAIRE properties of interest follow certain user-defined trends.

  - **Single trend checks**: check if a property extracted from a backend of choice satisfies some desired constraints over time. For example, the number of "harvested publications" from a content provider should be monotonic increasing with a max percent variation over time between two following values.
  - **Multiple trends checks**: check whether a relation among properties is held over time or not. For example, check if the increment of "H2020 OA publications" is greater than the increment of H2020 closed access publications during last month.

As we will see in Section 5.1.4, the metrics extracted from the four data components are many; it is important to notice that, since the four materialization processes run in parallel, the aforementioned metrics could be evaluated in different time, as soon as it is possible; hence to enable a correct comparison among them a synchronization routine should take place in order to align them to the same "epoch".

Finally, since the switch to public of the newly generated prepublic IS is performed manually, it would be valuable to investigate in a solution enabling to perform the switch automatically out of the results of controls defined over the extracted metrics.

2.1.5 Monitoring the provision data flow

Since monitoring intents have been outlined for each subsystem providing input to the provision data flows (i.e. the aggregation, deduplication and inference subsystems) as well for the publishing subsystem, which is located downstream, we deliberately decided not to outline monitoring intents for the provision data flow as it merely combine intermediate products, whose production is monitored (see Sections 2.1.1, 2.1.2, 2.1.3), into final products (pre-public IS), whose quality is being assessed (see Section 2.1.4).

2.2 The CORE data infrastructure

CORE (COnnecting REpositories) Data Infrastructure[18] aims to collect available OA (according to the BOAI[19] declaration) research outputs from repositories and journals. The initiative started focusing primarily on UK Open Access repositories, but it has recently started to gain momentum and expand across UK borders.

As seen for OpenAIRE, CORE DI is an application written in Java exposing services and user friendly Web User Interfaces (WebUIs) both to general public and to DI administrators and operation team. After a recent refactoring, CORE aggregation system has been re-engineered as a data flow (reported in 2.9) of different sequential processing components over data components. Each processing component is launched on a

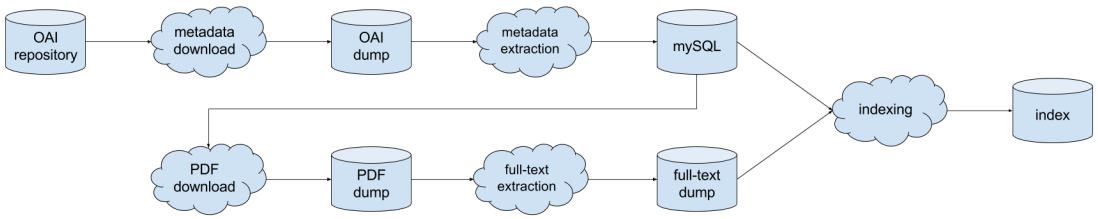


Figure 2.9: *CORE aggregation data flow*

per-repository basis and is devised for a specific task: *metadata download*, *metadata extraction*, *PDFs download*, *full-text extraction* and *indexing*. An orchestrator (called Supervisor) queues, schedules and instantiates different workers in order to trigger proper actions at the due time.

More specifically, the different activities that CORE features can be described as follows:

- **Metadata download**: a worker starts to process an OAI-PMH endpoint (a repository) and appends every result page in a file saved in a specific location on the filesystem. At the end of the metadata download process, the file contains a whole dump of the entire repository. In every moment, the filesystem stores the most recent "hard-copy" of all the repositories harvested so far;

- **Metadata extraction**: a worker takes one of these OAI dump files and process it record by record saving metadata information on a MySQL database;

- **PDFs download** starting from article metadata stored for a repository in the MySQL database, a worker processes the records one by one, and for each one of them attempts to locate and download the PDF of the article with increasingly sophisticate logic; whenever the download attempt fails it scales up (e.g. looking directly in the specific dc field, then looking for URLs pointing to valid PDFs with matching title, etc.). If every defined strategy fails, the worker gives up on that record and moves to the next;

[18]The CORE infrastructure, `https://core.ac.uk`
[19]Budapest Open Access Initiative, `http://www.budapestopenaccessinitiative.org`

- **Full-text extraction** a worker scans the downloaded PDF articles for a repository and tries to extract, when possible, the full-texts from each one of them. Whenever a full-text is found, the relative article metadata record is enriched on the MySQL database;
- **Indexing** the content present in the MySQL database for a given repository is indexed record by record into an Elasticsearch and becomes available for serving user queries coming from CORE web portal.

By analyzing the CORE data infrastructure [51], its design and its data flow, we identified some aspects that ought to be monitored over time and checked for consistency in order to gain insight of what is going on within the infrastructure.

Since the CORE infrastructure belongs to the same class of DI of OpenAIRE (i.e. literature aggregation DIs), some interesting aspects to be monitored are shared among the two DIs. However, because of the different design and implementation the use case served as a valuable support for designing the monitoring solution with an approach as generic as possible. The following aspects have been selected as subjects for our study:

- The average completeness of articles collected from a repository should cover at least the 80% of a pool of fields of interest agreed at infrastructural level.
- The number of articles collected from each repository should increase over time, as well as the number of PDFs downloaded and the number of full-texts extracted for each repository.
- The ratio between the number of successfully downloaded PDFs over the number of articles should be monotonically increasing over time. As the PDF download algorithm can be modified and improved, this indicator can play as a valuable aid to programmers and can drive a fine tuning and the design of a better PDF download process.
- The ratio of successfully extracted full-texts from PDFs over the number of available PDFs. This ratio should be monotonically increasing and over 0.98% success rate.
- Alignment of backends (for the sake of consistency):
  - the number of articles collected from OAI should match the number of articles extracted to mySql and indexed by Elasticsearch. If articles are lost on the way for any technical reason, an alert should be created;
  - the number of successfully downloaded PDFs in the file system should match the number of articles with PDF advertised by elasticsearch;
  - the number of full-texts successfully extracted from PDFs should match the number of articles with full-text advertised by Elasticsearch.

CHAPTER 3

State of the Art

The work in this thesis focuses on the challenges regarding Data Flow Quality Monitoring, in the general context of Data Infrastructures (DIs). Decades of experience and results in the area of monitoring systems have been taken into account, mainly in order to justify the need of this work, but also to re-use excellent scientific results whenever possible. The state of the art section revolves around the following research fields, considered relevant to the given problem: (i) *workflow quality and monitoring*, (ii) *data quality and monitoring*, and (iii) *data and workflow quality*.

As we shall see, scientific results in each of these topics are considerable, but, although such topics feature overlapping aspects and profound adjacencies, they are approached separately and rarely the designed solutions converge into "hybrid" frameworks, where aspects of quality and monitoring of data and workflows (i.e. processing) are blended to deliver the models and tools required by Data Flow Quality Monitoring. In this Chapter, we review the literature regarding the following research areas: in Section 3.1 we target the problem of workflow Quality of Service (QoS); while in Section 3.2 we will review some frameworks and techniques coping with data quality, a vast and challenging research topic on its own. In Section 3.3 we explore the solutions targeting the exploitation of concepts coming from data quality field into the context of workflows execution monitoring. Driven by the literature review, in Section 3.4 we provide a taxonomy characterising monitoring systems, be them for data or workflows, according to a number of desirable properties.

3.1 Workflow quality and monitoring

The concept of workflow, proper of business process modelling field [9, 44], is a purposely flexible concept, widely used in different fields of computer science to convey

the general idea of (manual or automated) orchestration of computational steps oriented to the processing of information, i.e. data. Relevant examples can be found in Service-Oriented computing [18, 79] and more recently in e-Science [41, 53, 66]. Indeed, computational and data-driven science in the third and fourth science paradigms has called for the combined use of modern computational technologies in simulations and knowledge discovery [45, 46].

With the expression *workflow monitoring*, the literature generally refers to the models, policies, and tools aiming to track (i) information relative to the real-time (and past) status of workflows (e.g. succeeded/committed workflows, ongoing tasks, failures and issues) and (ii) information relative to their execution (e.g. elapsed execution time for past workflows, progression) [44, 53, 66].

Workflow monitoring is often a built-in function, i.e. coupled with Workflow Management System (WfMS) that offer this functionality out-of-the-box. This the case for general-purpose WfMSs such as Airflow[1], capable of describing directed graphs of "jobs", i.e. execution steps, or for service-oriented WfMSs such as D-NET [57]. The same holds for Scientific Workflow Management Systems (SWfMSs) [53], which in several cases provide capabilities for monitoring the status of scientific workflow. Taverna [78], Galaxy [42], Kepler [3], Chiron [77], Askalon [34] all provide off-the-shelf native tools in this sense. *Scientific gateways*[2] – possibly based on such SWfMS – usually offer workflow monitoring capabilities in this sense too, e.g. WorkWays [76] and several others [28, 31, 32, 121].

A different approach is instead to provide workflow monitoring as a third-party capability, external to the WfMS at hand. In [47, 112], the authors designed Stampede, a generic monitoring infrastructure capable of monitoring the execution of complex (in general large-scale, parallelized and distributed) workflows. The Stampede approach relies on confluence and harmonization of log data collected from the remote execution nodes in which portion of the workflow is running to a centralized node for monitoring. Stampede enables real-time workflow failure prediction and analysis, and offers post-execution end-user tools for plots, statistics, analytics and programmatic access to data. The Stampede monitoring system has been initially integrated in Pegasus [30] SWfMS and later adapted to Triana [106] SWfMS, thus confirming the soundness and adaptability of the approach.

Workflow monitoring is considered one of the key aspects in SWfMSs, and more generally in WfMSs, but the most mature solutions in this ambit are mainly relative to the aspects of ensuring correct execution of the steps and optimization of resources. SWfMSs give life to discipline-specific data processing scenarios where workflow quality and measures of success are also driven by other factors.

Cardoso et al. in [26], pointed out that modeling and analysis of non-functional aspects of workflow systems is key for assessing and increasing their added value. The authors introduce the concept of *workflow quality* and pinpoint four quality axes: *specification*, *analysis*, *monitoring* and *control*. In particular, they focus on the specification aspect and define a quadri-dimensional model spanning over the dimensions of *time*, *cost*, *reliability*, and *fidelity*. *Time* is defined as the total time required for a task (a workflow activity) instance to process data fed as input into outputs. *Cost* is defined

[1] Airflow, `http://nerds.airbnb.com/airflow`

[2] Science Gateways, `http://sciencegateways.org/about/science-gateway-basics`

as the cost associated to the WfMS management and monitoring and to the runtime execution cost of workflow tasks. *Reliability* is defined as the ratio of the number of time a task reach the state "done" or "committed" over the number of times it reaches the state "failed" or "aborted". Finally, *Fidelity* is defined as a function of the effective design and refers to an intrinsic properties or expected characteristic of data products being created, transformed or analysed. The Fidelity dimension is intentionally highlighted to be the most problematic to define and hard to evaluate mainly because of the "fitness-for-purpose" effect, i.e. subjectivity of judgement and perception of the quality of data. In [27], the same authors propose a formal predictive theoretical workflow quality model based on time, cost and reliability dimensions, overlooking fidelity. They provide the formulae required to automatically compute quality metrics of workflows, and implement the proposed solution in METEOR WfMS[3].

Similarly, but focusing on SWfMSs, Gil et al. report on the results of a workshop on "Examining the Challenges of Scientific Workflows" [41] and advocate the need to provide frameworks and tools for specifying and monitoring quality of service requirements. The idea is to push beyond workflow execution time and bandwidth optimizations, hence to extend monitoring to other "quality" indicators, so as to cover other relevant aspects of workflow execution such as responsiveness, fault tolerance, security, and cost.

Although the need of monitoring workflow quality has been clearly stated in the literature and frameworks and taxonomies have been proposed, out-of-the-box tools for workflow quality monitoring are today missing. Data Flow Quality Monitoring cannot therefore be supported by adopting existing solutions in the area of SWfMSs and more generally WfMSs.

3.2 Data quality and monitoring

Literature about data quality is as old as data itself as it strives to understand to which extent data can be considered to have quality or not. An extensive literature review on data quality models is provided by Batini and Scannapieco in [15], from which it surfaces that the definition of data quality is non-trivial. Authors from different research communities (e.g. mathematics and statistics, management, digital libraries, computer science and engineering), hence with different background and target data or metadata [80, 96, 104], perceive the data quality issue from different angles, with different requirements and methodologies (intuitive, theoretical [115], empirical [120]). The definitions, classifications and frameworks so far produced are in several cases potentially relevant for Data Flow Quality Monitoring [11–13, 19, 22, 24, 37, 49, 62, 81, 85, 87, 92, 93, 96, 102, 103, 113, 115, 117, 118, 120].

In general, data quality is a multi-faceted problem which spans across different dimensions depending on the perspective and focus of the application domain. Known examples of data quality buzzwords in this ambit are accuracy, completeness, timeliness, precision, reliability, currency, relevancy, accessibility and interpretability. And for each of these data quality dimension multiple measurement methods can be found in the literature [15]. This diversity is not surprising as the notion of data itself may range from bit stored on disc, to data stored according to a given format, to data conforming to a conceptual information model, etc. Wang and Strong in [120] surveyed what data quality

[3]METEOR Workflow Management System, `http://knoesis.wright.edu/?q=projects/past_projects/meteor`

means from a data customer perspective and coined for the first time the expression *fitness for use* (also *fitness for purpose* [105]), meaning that *data quality* is met whenever "data are fit for the use by data consumers", a concept already anticipated in [6] in which data quality is defined as "a placeholder for whatever dimensions of data quality are relevant". Similarly, in 2008, ISO defined the standard ISO/IEC 25012:2008 in order to standardize data quality as "the degree to which the characteristics of data satisfy stated and implied needs when used under specified conditions" and provides "a general data quality model for data retained in a structured format within a computer system". It is useful however, when in the need of defining a custom notion of data quality, to refer to common taxonomies, which can drive data quality managers through a rigorous structured thinking and then be tailored to match local needs.

In terms of tools and frameworks, particular attention has been posed to (heterogeneous) Data Integration System (DIS) – the expression ETL (Extract-Transform-Load) is also found in the literature – and to the E-R database world resulting in a plethora of commercial and open source solutions (e.g. Attacama DQ Analyzer and DQ Center[4], Talend Open Studio for Data Quality[5], Data Cleaner[6], Data Monitoring Tool[7], Informatica Data Quality[8], Sourceforge's DataQuality tool[9] and Experian Pandora[10]), frameworks and research efforts [2, 10, 21, 63–65, 68, 75, 82, 83, 108, 116]. Such tools and frameworks mainly target *data profiling*, intended as the activity of analysing a data source to infer indices (e.g conformance, semantics, quality, interlinking) useful to enhance the reuse of the data source, and a few address data monitoring, intended as a repetitive (routinely) validation of the data to assess its quality over time. An extensive literature review on proposed frameworks and approaches can be found in [15].

An interesting data quality monitoring solution in DIS is the QBox architecture and tool, described in [33, 35, 43]. When integrating different external data sources, monitoring data quality becomes a major issue as the data sources involved are likely adopting data profiling activities specific to their technology, data model, and data quality models. Data profiling solutions are hardly interoperable, and coming up with a consistent data monitoring system, capable of checking quality constraints by verifying data sources ("local quality evaluation") and data integration status ("global quality evaluation"), is certainly far from being trivial. QBox is conceived as a system for assessing and monitoring quality of integrated data by enabling the integration of different data profiling systems. QBox mainly provides the following features:

1. a generic meta-model for the definition of quality goals and metrics,

2. a service-based infrastructure enabling the interoperability amongst different quality tools, and

3. an OLAP-based analysis and visualization dashboard.

[4] Ataccama, `https://www.ataccama.com/products/dq-analyzer`

[5] Talend Open Studio for Data Quality, `https://www.talend.com/download/talend-open-studio#t2`

[6] Data Cleaner, `https://datacleaner.org`

[7] Data monitoring tool, `http://www.uniserv.com/en/company/blog/detail/article/data-monitoring-tool`

[8] Informatica Data Quality, `https://www.informatica.com/products/data-quality/informatica-data-quality.html`

[9] Arrahtec "Open Source Data Quality and Profiling", `https://sourceforge.net/projects/dataquality`

[10] Experian Pandora, `https://www.edq.com/uk/solutions/experian-pandora`

The OLAP-based visualization dashboard offered by QBox supports the visualization of quality dimensions over time so that the user can check the outcome of subsequent ETL integrations and compare results over time.

QBox does not address the concept of data flow and the quality of Extract-Transform-Load process is intentionally declared out of the scope of the paper (here called Process Quality Scenario) [35]. The same aspects are missing in all data quality tools mentioned above, which tend to focus on the analysis of a data source rather than tracking its quality also in terms of how the data was generated. Accordingly, a tool like QBox could only partially address the challenges of Data Flow Quality Monitoring.

3.3 Use of data quality concepts in workflows monitoring

Although traditionally monitoring of quality for data and workflows have been following separate avenues, their research inheritance have been recently brought together in the field of e-Science scientific workflows (e.g. [1, 111]), where data and workflows give life to data flows demanding quality monitoring at the level of the data sources and processing. Poor quality of data, often collected from uncontrolled data sources and processed during an experiment, can yield revenue loss, bad resources allocation, wrong results and deductions, and dissemination or propagation of such subtle errors to subsequent experiments re-using previously produced data [44].

In [72, 86], Missier et al. propose an Information Quality (IQ) ontology enabling scientists and bioinformaticians to describe in scientific workflows their subjective perception of data quality requirements in a natural, yet formal and domain-specific, way. The ontology also fosters extension and reuse of already present quality concepts by means of a catalogue.

In [69–71], Missier et al. describe the Quality Views framework. Quality Views lets an user specify an abstract process that (i) collects Quality Evidence (QE) from a dataset in input (item per item) thanks to a Annotation Function (AF), (ii) applies one or more Quality Assertion (QA) functions to the computed QE in order to evaluate quality classes and partition the initial dataset into different quality classes, and (iii) associates quality actions to each quality class defined and perform that action to the corresponding data. This process enables, for example, the user to filter dataset according to multiple user-driven quality criteria. Quality views works under the assumption that AFs' implementations are provided, meaning that they are seen as black-boxes and are not exposed to the framework. The authors also provide a Haskel compiler for Quality Views and provide the Qurator[11] workbench implementing the Quality Views framework into Taverna[12] SWfMS. Qurator provides a compilation mechanism that translates the quality concepts modeled with Quality Views into the Taverna's Workflow Execution Plan (WEP), making workflows quality-aware. The study is supported by a case study in the protein identification field (proteomics). Indeed the approach is promising, but the implementation of Qurator for Taverna seems to be scarcely documented[13] and the project website broken.

[11] Qurator grant agreement GR/S67593/01, `http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=GR/S67593/01`

[12] Taverna SWfMS, `http://www.taverna.org.uk`

[13] Information quality in Taverna, `https://taverna.incubator.apache.org/introduction/taverna-in-use/information-quality`

Na'im et al. introduce in [74] an approach based on the instrumentation of Kepler SWfMS which enables the platform to provide visual cues – a three-colored blinking visual indicator on workflow execution dashboard – to the scientist running the experiment so that he can stop the execution if consider proper. The authors envisage also a rule engine to capture quality conditions and automatically react in accordance. The approach requires data quality values to be stored along with the data being processed. The interface provides a configuration window where it is possible to set three quality thresholds (i.e. *low*, *average* and *good*). As data is processed, the instrumented Kepler instance provides to show the visual clue about data quality on top of the workflow activities drawn within its execution dashboard. The solution is interesting, but it requires that the quality feature is a field of the data itself and must be housed within its data structure; a condition which cannot be guaranteed in general. Furthermore it provides data quality information within just a workflow execution, an assumption which could be a constraint in situations in which across-executions controls needs to be ensured. Finally it fails to provide aggregate views of quality in an entire dataset, as it focus only on the lowest granularity item composing the dataset. To the best of our knowledge, no further update has followed to the work here mentioned.

In [89–91], Reiter et al. propose a framework for the evaluation and monitoring of data quality during simulation workflow execution in order to detect promptly quality issues, control and steer efficiently the execution of the simulation and potentially prevent revenue loss [69] due to time loss in long-running experiments. They come up with a novel data quality measurement process for simulation workflows and describe how this approach and its components can be integrated into SWfMS in order to achieve quality-driven workflow execution. They identify three levels to drive workflows with data quality: workflow navigation (i.e. steering), service/algorithm selection and service/algorithm dynamic configuration. The authors support their findings with a case study on a Finite Element Method (FEM) simulation workflow which is able to steer during its execution according to quality parameters evaluated at runtime over the FEM grid used as input of a specific matrix solver.

In [55, 56, 101] Malaverri et al. propose and discuss and semi-automatic approach based on provenance enabling practitioners to assess data quality of artifacts produced during an experiment. They propose a provenance model compliant to the Open Provenance Model (OPM) [73] capable of recording, during the execution of a Taverna workflow, (i) generic properties about data processed and produced by the workflow, (ii) domain-specific properties evaluated by the application of metrics over data processed and produced by the workflow, and (iii) properties of different instantiations of a processing component within a workflow. Such properties are exported along provenance with the mechanism of "textual annotations" provided by Taverna, which however has to be instructed by domain experts. The authors then propose a three-steps methodology that outlines the basis for evaluating the quality of data components produced by an execution of a generic Taverna workflow. The study is supported by a domain-specific case study (agriculture research field).

In [97, 98], Silva et al. introduce an approach for the extraction at runtime of domain-specific attributes from intermediate and final raw data consumed and produced by running scientific experiments. Such extracted domain-specific data are then incorporated with provenance data and saved in the provenance database for later use. The

proposed provenance database stores both workflow specification (prospective provenance) presenting its structure in terms of activities and data dependencies and the workflow execution properties (retrospective provenance) and enables scientists to select and access relevant raw data, track them during the whole production data flow and assess whether they comply with defined quality criteria. The main advantage of the approach proposed is that attributes are extracted from raw data as they are generated instead of requiring to parse them *ex post*, once the workflow has ended. The study is supported with a case study comprising a Montage[14] large-scale scientific workflow (computational astronomy research field). As stated by the authors the work needs further generalization and extensions in order to be adopted broad-spectrum.

In [66], Mattoso et al. provide an extensive literature survey about workflow steering in High Processing Computing (HPC) scientific workflows and provides a taxonomy for concepts of workflow steering. The taxonomy includes data quality monitoring aspects as they are considered relevant to HPC SWfMS, but the study highlights that the issue still represent an open challenge in this context.

In summary, for several reasons, none of the approaches above is suitable for the kind of functionalities required by Data Flow Quality Monitoring, which targets DIs whose resources are not necessarily under the control of the DI quality managers and whose workflows may be only partly automated. Some of the approaches are based on assumptions that cannot be generally valid (e.g the ability of defining Annotation Functions at the item level), others are specific to given technological solutions or application fields (e.g. the ability to "steer" workflows based on given feedback, integration in Kepler) and cannot be adapted to different contexts.

3.4 A classification taxonomy for monitoring systems

The analysis of the literature above, and its validation with respect to the problem of Data Flow Quality Monitoring and its requirements, has led us to define a classification taxonomy of existing solutions in terms of a number of features characterising the methodology, strategy, and architecture of monitoring systems, across data, workflows, and data flows. The taxonomy grounds on the following terminology:

Target system It is the system to be monitored, e.g. data source, a WfMS, a SWfMS, Data Infrastructure data flows;

Monitoring system It is the system that monitors the target system, capable of routinely collect *metrics observations* over time and run validation tests;

Measurement (Metrics and Observations) A *metrics* is a named function (e.g. size of a data collection, impact of an algorithm over its in/out data) that produces *observations*, i.e. values, at given moment in time; observations relative to a metrics constitute the tangible description of the behaviour of data, process, or workflow to be observed; *measurement* is the action of generating an observation relative to a metrics;

Control It is a function that applies to the observations of one or more metrics and produces values representing the estimate of the overall quality of the entities under monitoring.

[14] Montage, `http://montage.ipac.caltech.edu`

The taxonomy develops across different dimensions, some of which are not totally independent, meaning that one choice in a dimension may affect available choices across others:

- *Monitoring system integration*: the degree of technological decoupling between target system and monitoring system, namely *embedded* versus *independent*;
- *Monitoring approach*: *blackbox monitoring* versus *whitebox monitoring*;
- *Monitoring exploitation*: the intended exploitation of monitoring control results, namely *human-driven* exploitation versus *machine-driven* exploitation;
- *Measurement customization strategy*: the flexibility of the monitoring systems with respect to the definition of own function metrics, namely *intuitive and out-of-the box* measurements and controls versus *developer-oriented customization* of measurements and controls;
- *Measurement location strategy*: the decision whether metrics observations are calculated at the target system or at the monitoring system site, namely *in situ* versus *remote* evaluation;
- *Observations storage strategy*: the decision whether data quality observations are to be stored together with data or separately, at the monitoring system;
- *Measurement and controls strategy*: the decision whether measurements constitute the minimal elements necessary to determine quality or instead a separate notion of controls, operating over input metrics, should be introduced;
- *Control strategy*: the decision to leave controls to machines or introduce humans in the loop;
- *Workflow awareness*: the ability of a quality model to capture the notion of workflows in its measurements, i.e. metrics and observations are contextualized to the execution of a workflow at a given time;
- *Workflow quality awareness*: the ability of a quality model to integrate notions of workflow (data and process) quality beyond mere performance parameters.

In the following we shall describe in detail such features, providing examples, and where needed commenting on their possible implementation in the context of DI data flows.

Monitoring system integration The monitoring function of a generic target system (in our case a DI) can be performed either *internally*, from a tool deployed within (*embedded*) the target system, or *externally*, by leveraging an external monitoring system pluggable on top of the target system and collecting measurements.

While the former solution is specially devised for the target system at hand and cannot be easily adapted to other contexts, the latter outlines a "companion" (*independent*) monitoring system which can be generally designed as:

- general purpose,
- programming language agnostic,

- technological platform agnostic,
- integrable at any moment in time,
- minimizing (possibly) the effort, i.e. cost, required to integrate with the target system.

The monitoring system takes away from the target system, thereby hiding the relative complexity, the monitoring layer by offering abstractions for the mechanism of communications, persistence of monitoring data, enforcement of controls and any other provided service (e.g. notification and alerting).

Monitoring approach In general, when dealing with monitoring, two possible monitoring strategies emerge: *blackbox* and *whitebox*, sketched in Figure 3.1. These two concepts are borrowed from system monitoring terminology[15] [110].

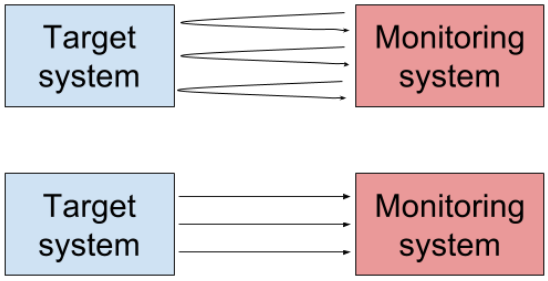


Figure 3.1: *Blackbox (top) VS whitebox monitoring (bottom)*

When a monitoring system implements a *blackbox* monitoring architecture, it means that it polls and pulls metrics out the target system, typically from public accessible services containing finished or intermediate data products produced by the workflow (e.g. filesystem, staging areas, databases), and typically when the workflow has executed (WfMS in idle). Despite a blackbox monitoring architecture requires no participation whatsoever from the target system, it is able just to capture external functionalities (i.e. end customers' view) and performs end-to-end checks, while the rest of the system's internals remains pretty much unobservable. The approach described in [33, 35, 43], for example, follows a blackbox strategy as it focus data quality measurements on pre-integration database (sources assessment) and on post-integration database (final product assessment).

As opposed to blackbox monitoring, a *whitebox* monitoring architecture requires instead a tighter cooperation from the target system which turns into an active counterpart in the monitoring architecture and provides more granular information about its internals that, with a blackbox approach, were simply unobservable. The user of the monitoring system needs only to take care of instrumenting the source code of the target system in the proper place in order to evaluate the metrics of interest. The solutions described in [74, 89], for example, follow this monitoring strategy. Furthermore, any typical workflow execution monitoring solution implements a whitebox strategy, e.g. for tracking execution times of workflow activities, i.e. the WfMS itself takes case of advertising this monitoring data (as this information would be lost otherwise).

[15] Aaron S. Joyner, presentation "Monitoring and Alerting", `http://www.ncsysadmin.org/meetings/1010/Monitoring_and_Alerting.pdf`

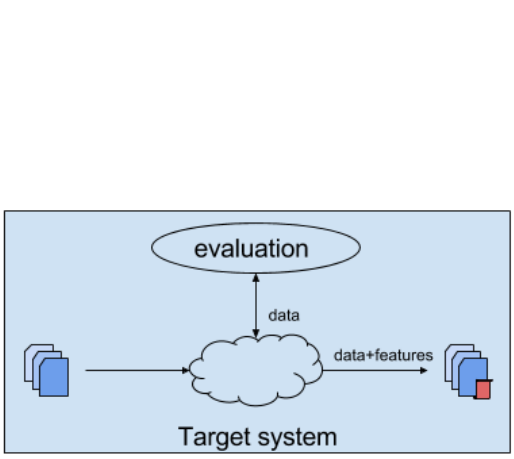
Monitoring exploitation The conclusions drawn by monitoring controls can be used with different purposes by humans, but also by machines. In particular, an *active* and a *passive* usages of quality notions have been identified:

- **Passive**:
  - *Reporting*: present a more or less organized view of quality metrics about the target system as in [33, 35, 43, 61, 74, 89, 90];
  - *Accountability*: show proof of system operations and internal state towards data sources, consumers and stakeholders in general [33, 35, 43, 55, 56, 61, 97, 98, 101].
- **Active**:
  - *Quality-driven steering*: use quality notions locally in order to steer the workflow, select or configure algorithms and services [89, 91];
  - *Quality-driven data manipulation*: use data quality notions locally in order to filter data out of shape (w.r.t. quality of data) and/or perform different actions (possibly corrective) on data according to data quality classes [69–71];
  - *Management*: use data quality notions in order to perform maintenance operations and/or trigger countermeasures in the target system [61] without altering the data content or without altering the workflow. For example, the quality information can be used in order to launch another workflow or in order to launch customized routines, e.g. temporarily graylist a data source whose data are out of shape.

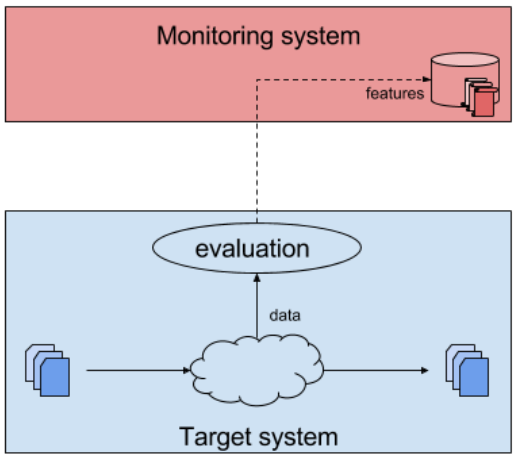
Measurement customization strategy A monitoring system can either provide out-of-the-shelf metrics implementations capable of working under certain controlled assumptions, or give total freedom to its users in order to implement (or customize existing implementations) their own solutions to their specific problems.

The first solution provides a *catalogue* of metrics implementations [89], while the second provides *placeholders* to be filled in. Metrics offered by the catalogue are specific to certain data types or backends and cannot operate under different circumstances. For example, the metrics implementation evaluating the completeness of a table in a E-R database cannot work on a full-text index; similarly, a metrics evaluating the signal-to-noise ratio for an X-Ray image cannot be applied on an audio file. Even more subtle, a metrics implementation working on a specific version of a backend might not work with another version of the same, or a metrics evaluating a property on data might be working only for a specific version of the data format/schema. Depending on the context, the literature provides a multitude of different measurement implementations [15].

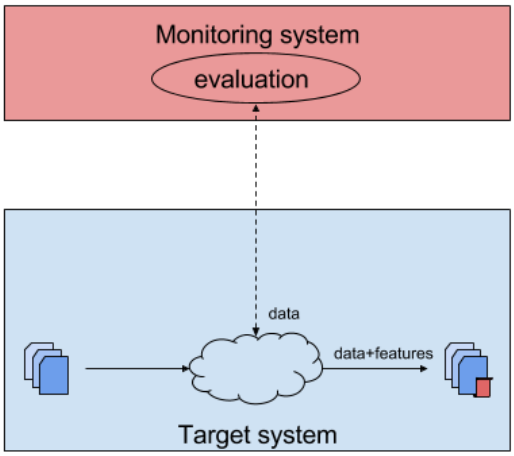
Because of this context dependency and since no assumption can be done *a priori* about data flowing in the infrastructure and the relative data quality, authors in the literature often refers to the "fitness for purpose" principle [105, 120] when referring to quality definition. For this reason, some other approaches also leave metrics implementation as a placeholder and empower the user to implement their own strategy [55, 56, 61, 69–71, 98, 101] which usually is domain specific.



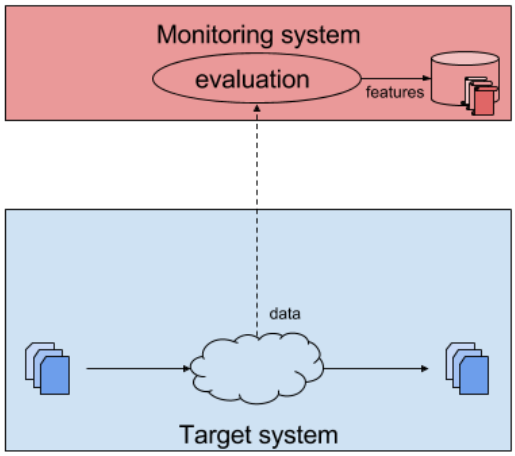
(a) *In situ evaluation of metrics, in band storage of extracted features*



(b) *In situ evaluation of metrics, remote storage of extracted features*



(c) *Remote evaluation of metrics, in band storage of extracted features*



(d) *Remote evaluation of metrics, remote storage of extracted features*

Figure 3.2: *Possible combinations of the available strategies for measurement location and observation storage*

Measurement location strategy Entities to be monitored can be "inspected" following two possible general approaches, shown in Figure 3.2: *in situ*, i.e. at target system site, or *remotely*, i.e. at the monitoring system site as shown in Figures 3.2a and 3.2b. Some work present in the literature adopt the *in situ* approach [61, 89, 90, 97], meaning for example that the evaluation of metrics over data is performed locally at the target infrastructure wherever the data is available.

Some monitoring frameworks present in the literature [89, 91] advocate for remotely evaluated metrics, a kind of metrics-as-a-service strategy, as shown in Figures 3.2c and 3.2d; of course in this case data have to be sent to the monitoring system. In order to cope with heavy data transfer, Reiter et al. in [89] conceived the possibility of passing data to their monitoring service by reference, a mechanism already used in [109] for quality-based data selection in Data-as-a-Service (DaaS).

Observations storage strategy Another direction of our taxonomy takes into consideration the way in which observations are stored and where. A possible approach is about housing such information *in band* (see Figures 3.2a and 3.2c), so that they are propagated along data, as a price tag or a fact-sheet essentially, in order to be used during the assessment of the quality of a data product [68, 117, 118]. For example, in Cooperative Integration Systems (CIS) [68], data sources partake to one scenario of data integration (rarely to different ones) and agree upon fostering the integration on how observations can be pre-calculated and passed over towards the global system embedded in the data source data.

The diametrically opposed approach instead is about storing observations *out of band* (see Figures 3.2b and 3.2d), separately from data, in a subsystem devised to persist and organize such information, typically located at the monitoring system site [55, 56, 61, 97, 98, 101].

The first solution is often far from trivial as it implies a tight collaboration among data sources and a great effort in order to integrate the modification proposed. This requirement is for example far from being considered as granted in DIs and relative data flows, since data sources are not directly controlled in the majority of the cases. In DI, there could be multiple workflows working over the same data, leading to an unbearable efforts in order to adapt and mediate data sources in all the use-cases in which is involved. In fact, the architectural modification suggested have to be adapted and implemented for every single scenario in which a given data source participates.

Measurement and controls strategy Another direction we explored in our taxonomy focuses on the degree of decoupling between measurements and controls. For example, metrics may yield a two-state (tri-state) output representing the success/failure (success/warning/failure) of the measurement. Somehow, in this scenario, controls are embedded in the notion of metrics, whose observations are already encoding an interpretation and an evaluation of the behaviour of the monitored entities. As suggested in [89, 90, 117], a symmetric approach is one where metrics have no judgemental value and are used to produce observations over "objective data quality features", e.g. completeness, conformance, accuracy, etc., and such observations be subject to the most diverse interpretations, encoded by means of controls, ultimately resulting in an indicator about "data goodness". For example, let us assume that the metrics $completeness$ maps data under examination into a measure ranging in $[0, 1]$. If the evaluation of such metrics returns 0.7, this "score" of the relative feature could be interpreted as acceptable in one context, but not in another. In this case, metrics are generic functions that, at a given time, can map data, processing steps, or data flows onto one or more observations representing features in the monitoring domain. The *interpretation* of such features is then subjective to the specific context, e.g. data flow, in which the features are being evaluated.

Control strategy Monitoring solutions typically contemplate two control application strategies, i.e. ways to assess quality given metrics and relative observations: *fully automatic*, i.e. machine-driven, or *Human-In-The-Loop (HITL)*, i.e. with ancillary support from an human operator. HITL-oriented solutions [55, 66, 74, 89, 90, 97]) foresee the

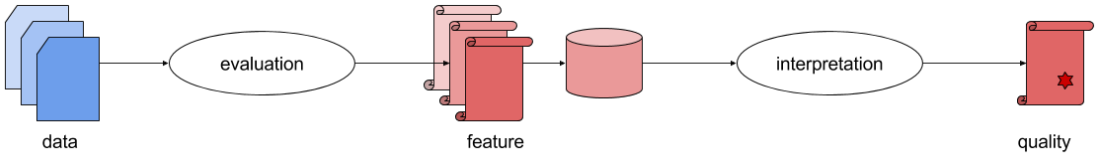


Figure 3.3: *Decoupling evaluation of data features from their interpretation*

possibility to:

- provide guidance and assist human operators (e.g. scientists or infrastructure managers) in assessing the quality of data produced [55, 56, 74, 97, 101];
- delegate quality interpretation to human operators by presenting the findings extrapolated by quality metrics and/or waiting for specific actions before proceeding in the workflow [66, 89, 90];

It is important to note that the introduction of a HITL brings an *alea* into the quality monitoring process as human beings evaluate in different ways according to their own knowledge and past experience, or think, feel and judge with different criteria over time [89].

Workflow awareness A monitoring approach can be more or less *workflow-aware*, meaning that observations are extracted being aware of the context (i.e. location in time and context of the workflow/data flow) where the measurement took place and that such information can be used in order to compare metrics and enforce controls overs them throughout time. A monitoring system is workflow *unaware* otherwise. To our knowledge, the majority of the approaches found in the literature tackle the monitoring problem without workflow awareness, meaning the just focus on one execution of the workflow. In [66, 89–91], for example, the data quality information is used locally in order to perform a choice (service/algorithm selection or configuration, workflow steering) within the same run. In [69–71] the WEP is instrumented with quality-aware workflows in order to perform different actions based on perceived quality of data, hence quality criteria are local to the quality workflow only. In [74], again the information about data quality is used locally to decorate the workflow representation in Kepler GUI. The only studied approaches that propose workflow aware solutions seems the ones described in [55, 56, 97, 98, 101].

Workflow quality awareness Several solutions in the literature focused on analysing workflow execution from a status and performance perspective (i.e. time elapsed for execution, number of tasks run, workflows and tasks executing, tasks failed, errors encountered, successfully committed workflows). Tools and services for workflow activity monitoring are often provided natively by several WfMS (e.g. D-Net [57], Airflow, Taverna [78], Kepler [3], Chiron [77], Askalon [34]). The Stampede monitoring infrastructure [47, 112] has been proposed in order to track workflow execution in complex and large-scale SWfMS such as Pegasus [30] and Triana [106] SWfMSs by mean of distributed data logs confluence.

Despite performance monitoring represent the first milestone when monitoring workflows, practitioners show a keen interest in monitoring more than just mere workflow

and WfMS performances, as mentioned in [26, 27, 41, 66].

Since in a DI the definition of a workflow can be in continuous evolution and refinement, and the data it has to cope with may evolve over time too, the "health status" of a workflow can be inferred also by runtime analysis of data as they go down the line and by enforcing controls over such data features. The works described in [55, 56, 69–71, 74, 89–91, 97, 98, 101] go along this research direction.

CHAPTER 4

MoniQ: A Data Flow Quality Monitoring System

Driven by the use-cases in Chapter 2 and the taxonomy defined in Section 3.4, in Section 4.1 of this chapter, we draw the requirements for Data Flow Quality Monitoring Systems (DFQMSs). Based on such assertions, in Section 4.2 we will define the architecture of the DFQMS MoniQ while in Section 4.3 and Section 4.4 we will define the *monitoring flow description language* and the *monitoring intent description language*, respectively. Finally, in Section 4.5, we conclude with an overall analysis of the integration effort required by MoniQ.

4.1 Requirements of Data Flow Quality Monitoring Systems

Given the use-cases discussed in Chapter 2 and the taxonomy introduced in Section 3.4, we can frame the rationale of the Data Flow Quality Monitoring Systems (DFQMSs) for Data Infrastructure (DI) data flows, of interest to this work. More specifically, for each of the monitoring systems features mentioned in the taxonomy, in the following we list and justify the preferred options for DFQMSs. These considerations will be served as key high-level requirements, summarized in Table 4.1, for the definition of the MoniQ framework, architecture and languages, in the next sections:

- *Monitoring system integration*: due to the heterogeneity of DIs and the likely autonomy of their processing components, DFQMSs should be designed to be as independent as possible from the target system at hand. Such systems must ensure some degrees of transparency with respect to development platforms, DI subsystems, data models, and data quality models. Most importantly, the adoption of DFQMSs should minimize the amount of development work necessary to integrate with the DI data and processing components.

- *Monitoring approach*: for the purposes of Data Flow Quality Monitoring, a black-box monitoring approach is simply not viable because of the evolutionary character and the dynamic nature of DI data flows and their components. By adopting a white-box monitoring architecture instead, the DI quality manager will be able to flexibly attach metrics and controls to data and processing components, and to their gluing data flows, to satisfy evolving control needs.

- *Monitoring exploitation*: DFQMSs do not impose any specific requirement on the side of monitoring exploitation and open up to both passive (accounting, reporting) and active (steering, management, etc.) exploitations.

- *Measurement customization strategy*: DFQMSs cannot make any assumptions on the DI components and data flows, their intent and goals, as well as their expected measures of correct behavior. As such, DFQMSs should be flexible enough to support DI quality managers at flexibly introduce custom metrics and controls that properly satisfy their needs. Of course this does not prevent DFQMSs to offer a "catalogue" of available metrics functions and relative implementations (e.g. for given back-ends or processing components).

- *Measurement location strategy*: DFQMSs must perform in-situ evaluation for a number of reasons. Beyond the decoupling requirements outlined above, which impose independence from data and processing components, in several application contexts, e.g. in a data-intensive workflow, the amount of data flowing in the DI can be massive and often there is literally not even space enough to save intermediate products into staging areas. In such a context, sending data through the network to the monitoring system for remote evaluation or, worse, remotely duplicating the data, is not an option. Passing data by reference as suggested in [89] for metrics evaluation, to the best of our knowledge, can be seldom practicable in DI context, since it requires persistence and globally reference-able data, constraints that in general cannot be guaranteed.

- *Observations storage strategy*: in accordance with the requirement of decoupling from the target system, DFQMSs cannot opt to alter the structure of data stored into data components or flowing through processing components as this option would reduce the application domain to a few selected DIs. DFQMSs must therefore offer storage for observations, which in turn are to be computed on the DI side.

- *Measurement and controls strategy*: DFQMSs should offer the maximum flexibility to DI quality managers, therefore deal with measurements as separate from control management.

- *Control strategy*: DFQMSs do not impose any specific requirement on the side of control monitoring, although in general monitoring and controlling a complex environment such as the one of DI data flows would benefit from a more scalable and fully automated approach.

- *Workflow awareness*: DFQMSs must be workflow aware, as their main intent is to track measurements in the context of DI data flows and over time.

Table 4.1: *DFQMSs requirements*

Feature	Requirement
Monitoring system integration	R1: DFQMSs are agnostic of DI component technologies
Monitoring approach	R2: DFQMSs follow a whitebox monitoring approach
Monitoring exploitation	–
Measurement customization strategy	R3: DFQMSs must allow custom definition and integration of metrics
Measurement location strategy	R4: DFQMSs must support in-situ evaluation
Observations storage strategy	R5: DFQMSs must offer storage for observations
Measurement and controls strategy	R6: DFQMS must support logical separation between measurements and controls
Control strategy	–
Workflow awareness	R7: DFQMSs must be workflow aware
Workflow quality awareness	R8: DFQMSs must support quality of workflows

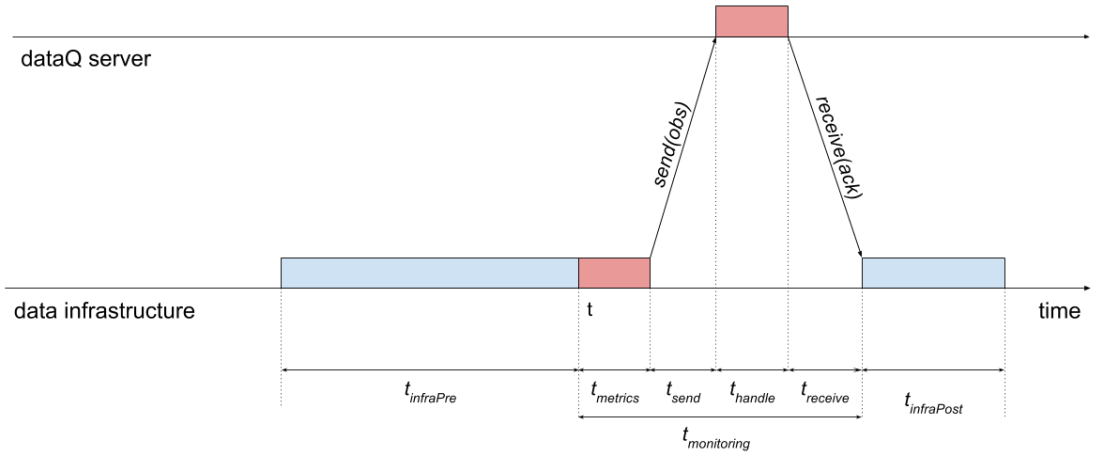


Figure 4.1: *Introduced latency due to MoniQ monitoring activities*

- *Workflow quality awareness*: DFQMSs must introduce a degree of workflow quality in order to satisfy the desire of DI quality managers to provide answers to data quality questions about a live system, in operation and in continuous evolution.

As a final comparison, in Table 4.2, we map the various approaches studied in the literature review presented in Chapter 3 to the taxonomy described in Section 3.4 and compare them with the requirements drawn for MoniQ.

It is worth noting that the drawn requirements imply the introduction of a overhead latency into the DI due to the metrics evaluation and the communication with the remote monitoring system. In Figure 4.1, we reported a simple scenario in which a metrics m starts to be evaluated by a sensor hook at time t, and outline the relevant time factors. The time employed for the normal execution of the data infrastructure $t_{infra} = t_{infraPre} + t_{infraPost}$ is represented in light blue, while in red we can see the overhead latency accountable to data flow quality monitoring. More precisely, the latency $t_{monitoring}$ introduced by the presence of MoniQ is equal to

$$\begin{aligned} t_{monitoring} &= t_{metrics} + t_{com} + t_{handle} = \\ &= t_{metrics} + t_{send} + t_{receive} + t_{handle} \end{aligned} \tag{4.1}$$

Table 4.2: *Comparison of MoniQ requirements against the state of the art*

	Workflow monitoring solutions		**Data quality monitoring solutions**	**Hybrid solutions**					**DFQMS**
Feature	WfMS/SWfMS	Stampede [47, 112]		Quality views [69–72, 86]	Na'im et al. [74]	Silva et al. [97, 98]	Malaverri et al. [55, 56, 101]	Reiter et al. [89–91]	MoniQ
Monitoring system integration	Internal, WfMS specific	External, dedicated (Pegasus, Triana)	External, backend specific	Internal, Taverna specific	Internal, Kepler specific	Internal, workflow specific	Internal, Taverna specific	Internal, SWfMS specific	External, general purpose
Monitoring approach	Whitebox (assess workflow performances)		Blackbox (assess backends)	Whitebox (compiled quality aware workflow)	Whitebox	Whitebox	Whitebox	Whitebox	Whitebox
Monitoring exploitation	Accounting and reporting	Accounting, reporting, failure forecast, analytics	Accounting and reporting	Quality-driven data manipulation	Visual reporting	Accounting, traceability	Accounting, traceability	Quality-driven workflow steering and configuration, reporting	Accounting, reporting. Trigger management actions, steering
Measurement customization strategy	None, typical metrics of workflow performances (exec time, success/failures, etc.)		Automatic/manual selection of available data quality metrics	User-defined data quality metrics and classes	Custom data features extraction	Custom data features extraction	Custom data features extraction and metrics over data	User-defined data quality metrics	User-defined data + processes quality metrics
Measurement location strategy	In-situ		In-situ or remote	In-situ	In-situ	In-situ	In-situ	In-situ and remote	In-situ
Observations storage strategy	Out-of-band	Out-of-band on Stampede dedicated repository	Generally out-of-band although some exceptions e.g. [68, 117, 118]	–	–	Out-of-band along with provenance	Out-of-band along with provenance	Out-of-band in a dedicated database	Out-of-band as time series
Measurement and controls strategy	–		Depending on the approach	Direct: quality metrics map to quality classes	Tri-state thresholds	–	–	Two phase interpretation	Two phase interpretation
Control strategy	–	Automatic and manual	Generally automatic	Automatic	Automatic	Manual: user queries	Manual: user queries	Automatic and manual	Automatic and manual
Workflow awareness	Aware		–	Unaware	Unaware	Aware	Aware	Unaware	Aware
Workflow quality awareness	Unsupported		Unsupported	Supported	Supported	Supported	Supported	Supported	Supported

The latency factors required in order to send and receive the observation to MoniQ server are indicated with t_{send} and $t_{receive}$ respectively. Finally, the overhead factor introduced by MoniQ server in order to handle and persist the observation is named as t_{handle}.

Under the desirable assumption that the overhead factor due to communication between the DI and the monitoring system t_{com}, and the t_{handle} latency introduced for handling the observation are as negligible as possible, the overall latency can be reduced to:

$$t_{monitoring} \approx t_{metrics} \tag{4.2}$$

which means that the dominant latency factor is the one due to metrics evaluation (i.e. its implementation), as pointed out in [89]. Therefore, the introduction of a monitoring layer affects the infrastructure "core business logic" execution time proportionally to the complexity of metrics implementation.

It is duty of quality managers to satisfy this condition as closely as possible during the implementation of the metrics of interest (see also Section 4.5 and Chapter 6). However, if the metrics implementation is optimized and does not require continuous access to external resources or calls to remote services (each of which is not recommended in any case), the monitoring overhead latency can still be contained. In the case externals resources may be required, it is advisable to prefetch and load them as internal resources of the sensor when it is instantiated.

In conclusion, thanks to the adoption of monitoring layer, dramatic time for troubleshooting, debugging and analyzing the DI status can be saved; thus we are confident that a trade-off between *introduced overhead* VS *benefits gained* can always be found in the majority of the cases.

4.2 MoniQ Architecture

MoniQ (pronounced "moh-NEEK") is a system supporting *quality managers* at monitoring their DI data flows. Its architecture, depicted in Figure 4.2, has been conceived as client and server architecture (requirement R1), where the server is the DFQMS and the client is the target system to be monitored, namely the DI and the relative data flows. MoniQ supports quality managers at:

- *Describing monitoring flows*: DI quality managers can express the semantic and the time ordering of their observational intents in terms of *monitoring flows*, which capture the essence of the DI data flows to be monitored, by means of the primitives of a *monitoring flow description language*; as we shall see in the following sections, MoniQ monitoring flows are not in general mapping directly onto the components of DI data flows, but rather representing their core parts to be monitored in terms of *storage blocks* (SBs) and *processing blocks* (PBs); for example, as illustrated in Figure 4.2, a MoniQ processing block may represent the processing effect obtained by combining multiple processing components in the DI; finally, monitoring flows are intended as ordered sequences of data and processing blocks, whose evaluation (i.e. the calculation of the relative metrics) takes place in a *block session*;

- *Describing monitoring intent*: DI quality managers can specify the metrics and controls required to perform the monitoring using the primitives of a *monitoring*

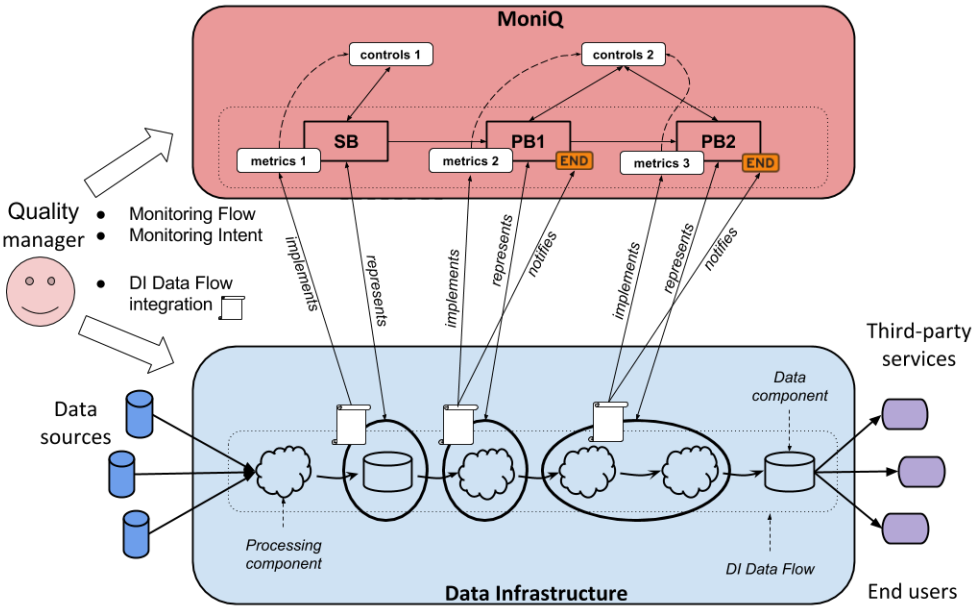


Figure 4.2: *MoniQ logical architecture*

intent description language; as we shall see in the following sections, metrics can be attached to individual blocks of monitoring flows as well as to (sub-parts of) such flows, hence controls can be at the level of individual blocks or such flows (requirement 8); not only, controls can be performed across multiple runs of the same monitoring flows (requirement R7), to evaluate metrics constraints over time; finally, the concepts of metrics and controls are kept independent (requirement R6), metrics and relative observations being the possible input of distinct controls;

- *Integrating DI data flows with MoniQ*: quality managers, on the client side of DI data flows, must code the programs needed to calculate the metrics and send them to the server side of MoniQ (requirement R2).

To this aim, MoniQ offers support for storing observations (requirement R5) that are produced at the DI side (requirement R4) by means of metrics implemented by the DI quality managers and specific to their measurement needs (requirement R3). Due to its decoupling from the target system, MoniQ is not equipped with pre-defined metrics, as the implementation of the relative functions depends on the technology, algorithms, data models, and development languages adopted on the DI data flow components. On the other hand, given specific DI platforms, it is possible for quality managers to realize and then equip MoniQ with toolkits of metrics implementations in order to make them reusable to other quality managers whose data flows are realized over the same technology.

4.3 The MoniQ Monitoring Flow Description Language

DI data flows span over a variety of different macroscopic aspects and tasks such as data collection from external sources, data processing (e.g. transformation, manipulation, integration, reconciliation, deduplication), data movement from/to different backends

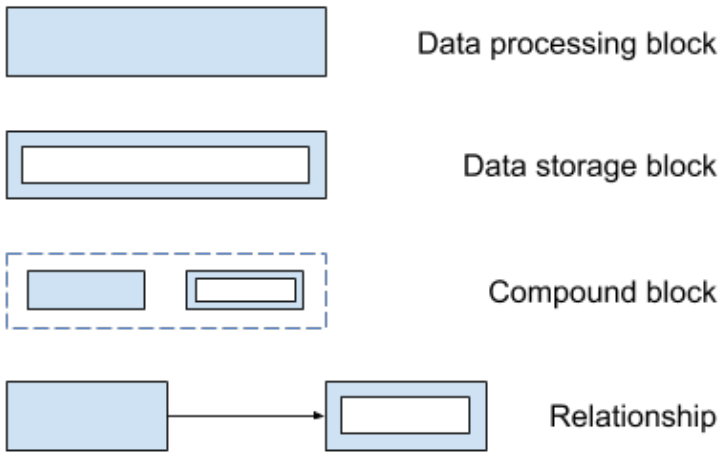


Figure 4.3: *DI construct blocks*

and data models, and data provision. They may include Human-In-The-Loop (HITL), in the sense some of their processing components are activated by humans (e.g. web applications, shell scripts) or be entirely managed via Workflow Management Systems (WfMSs). Data flows can be therefore specified in a natural language or given human protocol (e.g. Protocols.io[1]), for humans to read and execute, or be encoded in any of the workflow languages available in the literature, for machines to orchestrate. In any case, DI data flows describe sequences of actions, modelled by processing components, over data components.

The MoniQ monitoring flow description language is graphical, as it is intended to reflect the impact of a graphical user-interface, and takes inspiration from well known graphical languages for data flow representation. Such languages are built on the analogy between systems for data processing, e.g. Data Infrastructures, and "production systems" [6, 49, 83, 95, 119]: digital data is the "raw material" or the "final product", data processing is the manufacturing process, and data flows are the "assembly line". Such languages typically capture the notions of data sources, data storage, data processing, and data consumers and describe the relationships between components in a data flow. The monitoring flow language is instead intended to capture the data flow monitoring logic: how areas of the DI, represented by monitoring blocks, are to be supervised (i.e. behaviour observations are to be sampled) and in which specific time-ordering, by MoniQ.

The monitoring description language models *monitoring flows* as workflows built on three kinds of blocks: *data storage blocks*, *data processing blocks*, and *compound blocks* using the formalism depicted in Figure 4.3. A block B is concerned with tracking behaviour of a data flow component and collecting quality *observation* sent by the DI data flows to MoniQ during their execution. Monitoring flows can combine blocks via directed relationships dictating the expected ordering of the relative observations. More specifically, if blocks B_1 and B_2 are in a relationship $B_1 \rightarrow B_2$, then the measurement session of B_1 needs to be completed, i.e. all observations from B_1 must be acquired, before the measurements of B_2 can start.

As mentioned and exemplified in the previous section, monitoring flows aim at

[1] Protocols.io, `https://www.protocols.io`

modeling the high-level blocks of DI data flows to be monitored. As such, its data or processing blocks are not necessarily meant to map directly onto the DI components as well as monitoring flows are not necessarily mapping onto the full business logic of DI data flows; e.g. quality managers may be interested in monitoring a sub-part of one of their DI data flow. Indeed, it is up to the quality manager to decide what a data storage block has to represent, in such a way his monitoring needs are satisfied. Blocks can represent only components (or combinations of them) under the jurisdiction of the quality manager, meaning that quality managers must be able to embed sensors implementations close to their components.

Data storage blocks A data storage block represents the intention of modeling a data source component or, in more complex scenarios, a combination of data source components, logically seen as one storage system. For example, a filesystem providing access to the content of a file or a folder in a specific location, a database exposing to data stored in a table, or a NoSQL store (e.g. MongoDB) providing access to a collection of documents. In general, it is assumed that data storage blocks will be concerned with metrics and observations relative to collections of objects rather than to the individual objects, that is the underlying DI components typically support APIs to implement such metrics. For example, an API enabling to perform aggregated observations over persisted collections such as counts of certain classes of items, number of occurrences of a given property, number of empty fields of a E-R column, number of files in a folder and so on.

Definition 4.1. A *data storage block*, here also referred to as SB, represents the intention of monitoring a data management area in the DI supporting access to one or more *data collections*. In particular, the following features apply:

- A storage block starts when the first observations relative to the block reaches MoniQ;
- The start of a block spawns a new block session;
- The block session terminates correctly when all metrics have been measured as specified by the quality manager.
- The block session terminates incorrectly when one of the metrics is received twice before the correct termination of the session.

□

Data processing blocks A data processing block models the intention of monitoring a computing function, which can be made of a pipeline or combination of processing components, but it is perceived as one action to be executed over data and/or to produce data and to be monitored. Examples of processing blocks to be monitored can be a mediation component in the DI, entitled to collect data from an external data source to deposit them in a data component in the DI. In this case there is not input to the component, which generates instead a data collection as output. In a more general sense, a processing block may represent combinations of DI processing components,

whose goal and machinery are known only to the quality manager who is interested in monitoring the output of the overall processing block starting from a given input.

More specifically, a data processing block is intended to model monitoring of a common computing scheme in data flows which is that of processing a sequence of input data to produce a sequence of output data; e.g a *for-each*, *while*, *parallel* algorithmic skeleton. This general skeleton is typical to processing components in any DIs, where for example a result set of records is passed over to a service in order to execute a transformation to each of the records (multiple input multiple output, MIMO), or a collection of objects is processed to return one "fusion" of such objects (multiple input single output, MISO). By introducing this level of granularity, still without entering the representation of input and output data, the MoniQ framework allows the definition of *sub-process metrics* and observations at the level of the individual sub-calls of the processing logic behind a processing block; e.g. with regard to the MIMO example above, the "quality" measure of the improvement of one record before and after the transformation. This is of crucial importance, since the metrics relative to a processing block depend on combination or aggregations of observations relative to sub-process metrics; e.g. with regard to the MIMO example above, the average quality impact of a processing logic after its execution, hence after all its sub-calls have been executed. To this aim, MoniQ offers storage for the observations relative to the individual sub-calls while providing support for the calculation of *process metrics*, obtained by properly combining the observations produced by sub-process metrics. Without this abstraction mechanism, the modeling of the same metrics would require the quality manager to develop a temporal cache for the intermediate observations (sub-process metrics), conflicting with requirement R5. Obviously, we cannot pretend to have modeled all possible patterns of processing logic. Some monitoring use-cases may well escape the framework model and possibly require complex implementations of metrics functions on the DI side. However, we are confident that the most common use-cases can be expressed with such abstractions and therefore benefit from any MoniQ framework implementation.

Definition 4.2. A *data processing block*, here also referred to as PB, represents the intention to monitor a generic processing logic $\mathcal{F}$ that is run k times within the execution of a DI data flow, according to a sequential or parallel computing pattern.
In particular, the following features apply:

- A processing block starts when the first observation relative to the first invocation of $\mathcal{F}$ reaches MoniQ;
- The start of a block spawns a new block session;
- The block session correctly terminates with an explicit *session closing* notification;
- If a closing notification is not sent, the session may remain "idle"; the MoniQ framework allows to set time-to-wait thresholds, after which the session is terminated incorrectly;
- When the block session terminates, all process metrics observations are calculated by MoniQ.

□

Compound blocks For monitoring purposes, blocks can be grouped into so called *compound blocks*. Such blocks are used to model the fact that the including blocks can be seen as part of the same session, even though their metrics are measured at different times, following the ordering specified by the monitoring flow. This allows metrics of different blocks to be aligned at the end of the compound block session, in order to calculate *compound metrics* or perform controls that require such metrics to be time-aligned.

Definition 4.3. A *compound block*, here also referred to as CB, represents the intention to monitor the overall behaviour of a group of blocks. CB can include a subset of data, processing, and compound blocks that are part of a monitoring flow; its effect is to inherit the metrics of the blocks it contains. If CB includes blocks $B_1, \ldots, B_k$ then the following features apply:

- A compound block starts when the first B_i starts;
- The start of a compound block spawns a new block session;
- The block session correctly terminates if $B_1, \ldots, B_k$ correctly terminate;
- The block session incorrectly terminates if any B_i incorrectly terminates.
- When the block session terminates, all compound metrics observations are calculated by MoniQ considering the metrics of $B_1, \ldots, B_k$ as aligned at session closing time.

□

The need of monitoring an entire monitoring flow, hence of defining metrics or controls that refer to the scope of the whole flow, is modeled with a compound block including all blocks of the monitoring flow.

Monitoring flows In order to capture the notion of data flow quality monitoring debated in this thesis, MoniQ defines a monitoring flow (*m-flow* as a directed acyclic graph (DAG) of data storage blocks, processing blocks, and compound blocks; see Figure 4.4 for an example. MoniQ provides quality managers with the abstractions required to express which blocks they are willing to monitor and, most importantly, how such blocks are chronologically related and can be monitored altogether. Specifically, a monitoring flow:

1. Defines the temporal ordering of block sessions relative to the blocks: indirectly this defines the temporal ordering in which the metrics relative to blocks should be measured;
2. Defines the temporal scope of an *m-flow session*: the start and termination of all session blocks in an m-flow reflects a complete execution of the DI data flow to be monitored; multiple executions, sequential or parallel, of the data flow therefore correspond to different m-flow sessions, with relative start and termination time.

Figure 4.4: *Monitoring flow example*

Definition 4.4. A *monitoring flow* (*m-flow*) is represented as a DAG of possibly interconnected processing blocks, possibly alternated by storage blocks, possibly including compound blocks. In particular, the following features apply:

- An m-flow starts when the first block in the flow starts;
- When a m-flow starts a new *m-flow session* is spawned;
- The m-flow session *correctly* terminates when all sessions relative to the blocks in the data flow have correctly terminated, respecting the proper ordering imposed by the DAG, if provided;
- The m-flow session *incorrectly* terminates when one of the block sessions is restarted before the m-flow session has correctly terminated; if the block session restarted does not violate the monitoring flow specification, then a new m-flow session is started.

□

The semantics of termination of an m-flow session is driven by the intuition that all metrics observations that will be attached to the blocks of the monitoring flow are assigned to the scope of an m-flow session. If the same metrics is computed again within the same m-flow session, before the m-flow session has correctly ended (all its data and processing components have produced their metrics and in respect of the ordering), it means that the monitoring flow has somehow failed to comply to its specification. In addition, if possible, MoniQ reacts by spawning a new m-flow session, as if the monitoring flow had started again. From the monitoring perspective, MoniQ keeps track of all m-flow sessions, and relative block sessions. The incorrect and correct terminations allow to investigate possible misbehaviours while being tolerant to temporary failures of the DI.

4.3.1 Monitoring flow examples

As explained above the quality manager in the need of monitoring a set of DI data flows using MoniQ will have to specify the monitoring flows, the relative blocks and metrics, and then implement the thin integration layer between these and the DI components. The implementation work is left to the minimum, with MoniQ exposing the APIs to be invoked in order to accept observations and send process closing notifications. In this section we report two examples of possible monitoring scenario of a data flow in a "staged data infrastructure" and in a "streaming data infrastructure".

Staged data infrastructure The staged infrastructure data flow illustrated in Figure 4.5 is typical of Aggregative Data Infrastructure (ADI) [7, 61]. It describes a data flow where a processing component "harvests" a collection of XML metadata records from

an institutional repository data source via OAI-PMH standard protocol and stores the collection in a MongoDB data component. Another processing components collects the records and applies an XML transformation function that improves the quality of the records in order to make them conforming to a different XML semantics and structure. The result of the transformation is stored in another MongoDB data component. The quality manager is interested in monitoring two data flow quality aspects over time: the average impact of the transformation function to improve the data flow, the increasing level of conformance of the original repository. The level of conformance of a record to the target schema can be calculated via a metrics $conformance$, which given an XML record returns a real in the range $[0 \dots 1]$. To this aim the quality manager defines a monitoring flow that includes a processing block $CollectionBlock$ relative to the record harvesting and storage into the MongoDB component, and a processing block $TransformationBlock$ relative to the transformation component. In MoniQ the quality manager defines (i) a process metrics $collectionConformance$ for $CollectionBlock$, which is the average of the sub-process metrics ***originalRecordConformance***, and (ii) a process metrics $overallImpact$ for $TransformationBlock$, which is the average of the observations relative to a sub-process metrics $recordImpact$. Finally, the quality manager defines a control at the block level, which verifies that $collectionConformance$ has an increasing trend over time and visualizes $overallImpact$ over time, and, by introducing a compound block, defines a control at monitoring flow level, which verifies the average ability of the transformation to improve the original collection is high (beyond a given threshold) when the quality of the original collection is low (below a given threshold). In order to instrument these metrics, the quality manager needs to code a thin integration layer, which implements the metrics, sends the observations, and sends the closing notifications when needed. Specifically, it will need to:

- Embedding in the record harvesting component code to calculate the function $conformance$ over the individual XML records collected and stored; the code should then invoke the MoniQ API to send an observation:

$$\langle originalRecordConformance, conformance(XMLrecord) \rangle_{CollBlock} \tag{4.3}$$

- Embedding code to invoke the MoniQ APIs to send a closing notification for the block $CollectionBlock$ whenever the harvesting component in the DI has concluded;

- Embedding in the record transformation component code to calculate the function

$$\begin{aligned} &diff\,(outputXMLrecord, inputXMLrecord) = \\ &conformance(outputXMLrecord) - conformance(inputXMLrecord) \end{aligned} \tag{4.4}$$

  after every transformation of an input record; the code should then invoke the MoniQ API to send an observation:

$$\langle recordImpact, diff\,(outputXMLrecord, inputXMLrecord) \rangle_{TransfBlock} \tag{4.5}$$

- Embedding code to invoke the MoniQ APIs to send a closing notification for the block $TransformationBlock$ whenever the harvesting component in the DI has concluded.

The integration cost is minimal, while MoniQ provides the scaffolding needed to keep track of the observations over time and run the required controls, which can in turn be modified, removed or added to satisfy future monitoring needs, still benefiting from the history of quality observations persisted by the system.

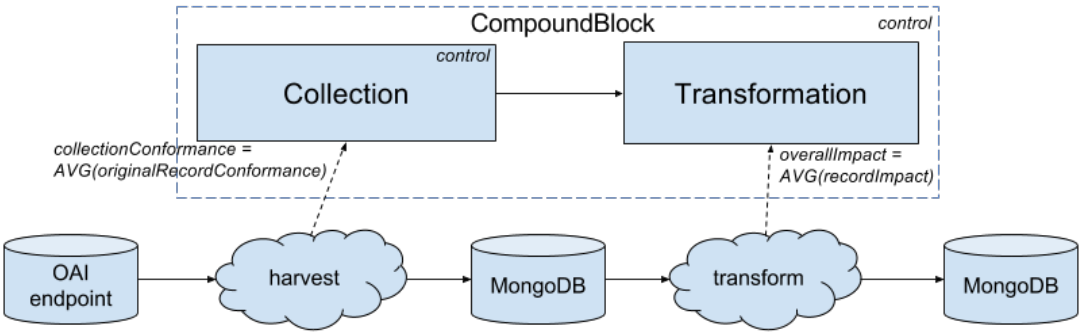


Figure 4.5: *Example: literature aggregation data flow*

Streaming data infrastructure The streaming infrastructure data flow depicted in Figure 4.6 describes a processing component collecting Twitter data,[2] a subsequent component tagging the tweets according to a strategy that is supposed to improve over time, and a final component storing the tweets in a data component (a MongoDB) only if they respect a predicate on the enriching tags. As an effect of improving the strategy, the tweets to be stored are supposed to grow. To monitor and control this trend, the monitoring flow in MoniQ is designed to keep the daily ratio of incoming tweets and stored tweets. The first processing block represents the collection of tweets, where every invocation of the $\mathcal{F}$ returns a metrics $call$ with observation "1"; the block has a process metrics $DailyTweetsCollected$ that is calculated as the sum of all observations relative to $call$. The second block is a storage block with a metrics $DailyTweetsStored$. The m-flow has a control at the level of the block, which checks that over time, i.e. across multiple workflow sessions, the ratio between the two metrics $DailyTweetsCollected$ and $DailyTweetsStored$ decreases. In order to implement this scenario, the DI quality manager has to:

- Embed in the component collecting from Twitter APIs a line of code to invoke the MoniQ APIs in order to send the observation $\langle call, 1\rangle$ relative to PB every time a tweet is collected;
- Set via MoniQ user interfaces the process metrics $DailyTweetsCollected$ as the sum of the observations of $call$ taken in the process session;
- Implement the metrics $DailyTweetsStored$, which calculates the total of tweets stored in MongoDB during the day;
- Implement in the DI a process that every day sends a closing notification to PB and subsequently fires the calculation of the metrics $DailyTweetsStored$.

[2]Twitter streaming API, `https://dev.twitter.com/streaming/overview`

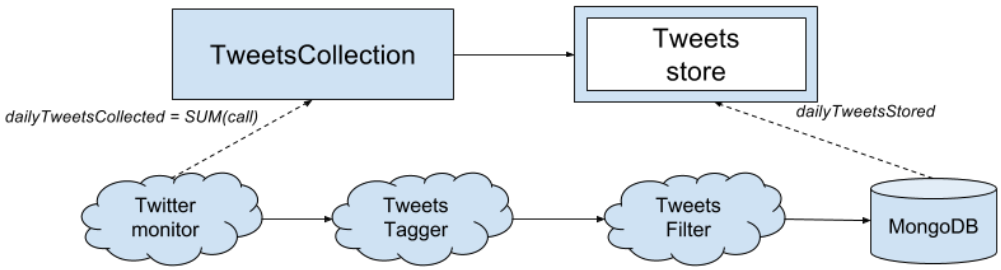


Figure 4.6: *Twitter streaming infrastructure*

4.4 The MoniQ Monitoring Intent Description Language

Once the DI's selected data flow has been modeled as a corresponding MoniQ m-flow, in accordance with the formalism introduced in Section 4.3, the DI quality manager can start specifying its monitoring-related intents. In the previous sections we have already used the terms *measurement*, intended the evaluation of a *metrics* producing an *observation*, and the term *control*, intended as a predicate over metrics observations, whose failure raises a warning to the DI quality manager. These concepts were intuitively introduced to better explain the monitoring flow description language and understand its nature. In the following, we shall define the details of the *monitoring intent language*, describing its expressiveness and monitoring ability. An overview of how monitoring intent concepts and monitoring flow concept are related to each other can be found in Figure 4.7; the concepts proper of MoniQ are coloured in red, while the concepts proper of the monitored DI are coloured in blue.

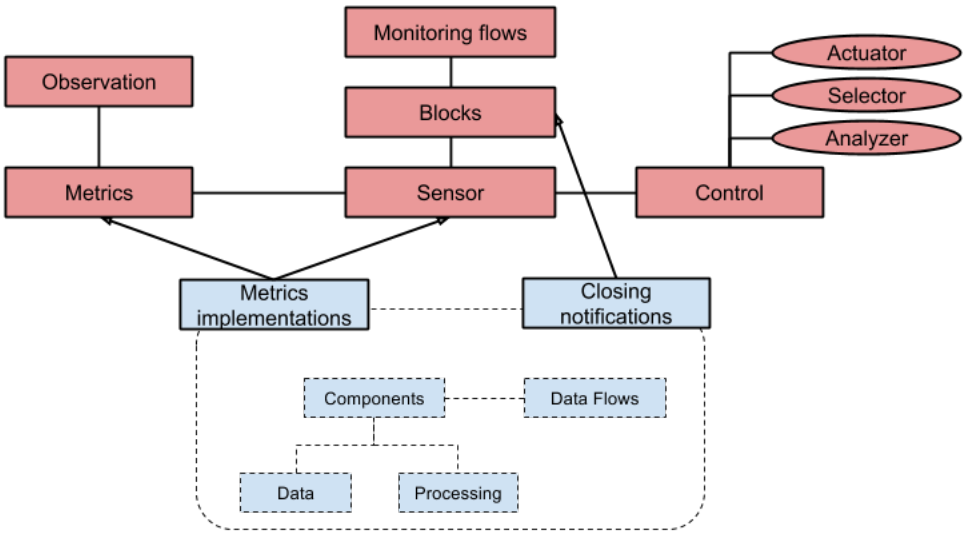


Figure 4.7: *MoniQ's information model*

A *sensor* is intended to provide a scope of metrics, observations, and controls for a given block. A block may have different sensors and each sensor declares at least one metrics, i.e. groups them in a context. Vice versa, each metrics is associated to one or

more sensors, and a sensor is associated to one block. Controls are associated to one sensor and can therefore refer only to all metrics included in the sensor scope.

4.4.1 Metrics

Metrics can be about everything the DI quality manager would like to monitor and are specific to the DI application context. They can be about data collections, data objects, or processes, they can measure execution time, expressions of quality, or size. MoniQ defines two classes of metrics: *basic metrics*, which are metrics whose observations are calculated in the DI and sent to the framework, and *derived*, which are those calculated by the framework.

Definition 4.5. A *basic metrics* m is a generic function expressing a feature of the DI to be measured as a value v (not to be confused with an observation, to be described below):

- *Data metrics* dm apply to storage blocks and are intended to measure features of data managed by DI;
- *Sub-process metrics* spm apply to processing blocks and are intended to measure features of a processing function $\mathcal{F}$ part of the DI data flow during the execution; sub-process metrics do not surface at the level of the processing block, in the sense controls cannot be performed over their observations.

The implementation of data and sub-process metrics resides in the DI. It is therefore the DI that calculates and sends to MoniQ the relative observation, in respect of the m-flow. □

Definition 4.6. A *derived metrics* is obtained by other metrics, be them basic or derived and is of two kinds:

- *Process metrics* pm apply to processing blocks and are intended to measure features of the application of the processing function $\mathcal{F}$ in a block session; as such they are obtained as aggregation functions over sub-process metrics in the same block;
- *Compound metrics* cm are intended to measure features that depend on the metrics of the sensors included in the block; as such they are obtained as aggregation/composition functions over such metrics.

The implementation of process and compound metrics resides in MoniQ, which includes a selection of aggregation functions (e.g. count, sum, average, etc.) that are executed on the server side over the observations relative to the metrics involved. MoniQ allows quality managers to define and integrate custom metrics functions. □

For example, a data metrics could take into account the amount of $NULL$ fields in a E-R database column, the number of files in a filesystem path, the size of a folder or the number of records persisted in an index matching a certain query.

For example, a sub-process metrics could evaluate the size of data in input to a process block, or the compliance to a given standard of reference of data produced in output, or again the completeness of data in input against a certain schema defined as:

$$m_{completeness} = \frac{N_{initializedFields}}{N_{mandatoryFields}} \tag{4.6}$$

For example, a process metrics can extract the execution time of a given call of $\mathcal{F}$, its heartbeat (meaning just the event of the call itself) or the normalized execution time evaluated for example as

$$m_{normExec} = \frac{t_{end} - t_{start}}{\sum_{i=1}^{I} size(data_{in})} \tag{4.7}$$

or how the $\mathcal{F}$ performs w.r.t. the input and the output, e.g.

$$m_{enhancement} = g(data_{out}) - g(data_{in}) \tag{4.8}$$

Hence, although the semantic of a metrics is described by its type (i.e. data, sub-process, process, and compound), the implementation of a metrics must respect the specification of its type and is left to the quality manager so that he can fit the specific use case and context of application.

A metrics is peculiar to the data type of the data under exam, hence a metrics measuring a certain feature has different implementations for each *data type* it has to work on. For example, the implementation of the metrics *completeness* working on an XML record is different from the implementation of completeness for a row from a CSV file. An extensive literature review about the plethora of different techniques for data quality metrics evaluation in different fields can be found in [14, 15].

As a rule of thumb, a metrics "core business logic" should be "self-contained" and never rely on access to external data or invocation of remote services, as it is advisable to perform evaluations that return as quick as possible. However, it is always allowed to rely on remote invocation in order to externalize part of the business logic to third-party services. Furthermore, whenever possible, it is advisable not to repeat part of the job already done by the DI "main business logic". For example, if a file is already open and loaded into memory by the infrastructure, it is advisable not to repeat the same within the metrics implementation.

4.4.2 Sensors

Metrics are the means for tracking observations of features relative to the DI data flows. For this reason, they are associated to a block, which is a placeholder for observing data and process in the DI. MoniQ introduces the concept of *sensor* intended as a "related" set of metrics, together responding to the same set of controls.

Definition 4.7. Given a block of reference B, be it a data processing or a data storage block, a *sensor* s for B specifies one set of metrics for B, together with the set of controls to be applied over these metrics. In particular:

- A sensor s anchored to a data storage block is constituted by a set of data metrics DM, a set of compound metrics CM derived from metrics in DM, and a set of controls C:
$$s_B = \langle DM, CM, C \rangle \tag{4.9}$$
- A sensor s anchored to a processing block is constituted by a set of sub-process metrics SPM, a set of process metrics PM obtained from metrics in SPM, a set of compound metrics CM derived from metrics in PM, and a set of controls C:
$$s_B = \langle SPM, PM, CM, C \rangle \tag{4.10}$$

Figure 4.8: *Graphic representation of a sensor*

- A sensor s anchored to a compound block including $B_1, \ldots, B_k$ is constituted by a set of sensor metrics SM selected from the sensors of B_i, a set of compound metrics CM derived from metrics in SM, and a set of controls C:

$$s_B = \langle SM, CM, C \rangle \tag{4.11}$$

One block B may have one or more sensors s_B, which are graphically represented as the circles reported in Figure 4.8. □

On the DI data flow side, the implementation of basic metrics m relative to a sensor s_B must therefore mark its measure v with s_B, in order to define the precise scope of the measurement. To this aim, during a block session, a metrics m generates *observations* o, defined as follows:

Definition 4.8. An *observation* o for m is an object that contains the value v returned at time t by the metrics m from the sensor s_B, plus some contextual metadata map about the measurement, derived from the DI context of the metrics. An observation is described as a tuple of the form:

$$o = \langle t, m, v, s_B, map \rangle \tag{4.12}$$

where map contains a set of observation attributes, represented as set of label-value pairs $map = \{\langle l_1, v_1 \rangle, \ldots, \langle l_k, v_k \rangle\}$. We define:

- O_m as the set of observations relative to m;
- $o_{ss_B,m}$ as the (only) observation generated for metrics m in the session ss of block B; metrics can generate only one observation in each session.

□

The attributes in map are introduced to instrument an observation with metadata about the context of involved block, to be used for metrics investigations, define accurate controls, and introduce different monitoring axes. For example, let us consider the aggregation data flow described in Figure 4.5 and the relative m-flow. Its formal definition, after the introduction of sensors, would be the one in Figure 4.9. Let us suppose that the data flow executes weekly the same processing steps, possibly in parallel, over around 700 data sources DS_i. Each execution regards a different data source, where metrics implementations will calculate observations like:

$$\langle t, collectionConformance, 0.5, collectionSensor_{CollectionBlock}, \emptyset \rangle \tag{4.13}$$

Such measurements are ambiguous and regard different DI scopes, identified by different execution instances of the same data flow. An optimal strategy is to use labels in map in order to characterize observations by the data source of interest. In this case

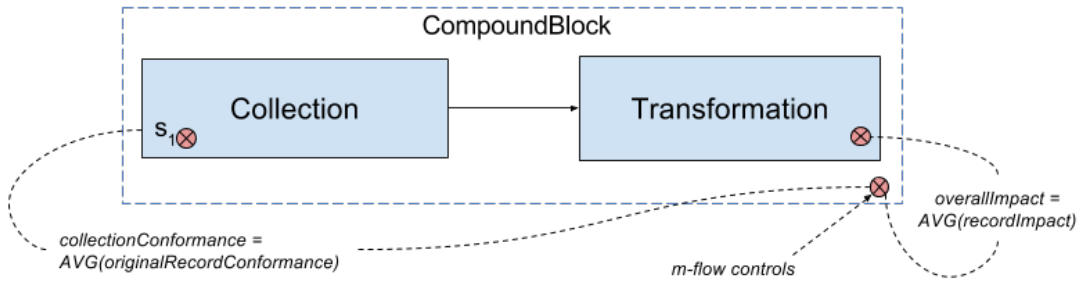


Figure 4.9: *Aggregation data flow with sensors*

the metrics function, taking into account its current execution context, can enrich the observation it produces with disambiguating information, for example a unique reference to the data source target of the data flow execution:

$$\Big\langle t, collectionConformance, 0.5, collectionSensor_{CollectionBlock}, \big\{\langle source, DS_i\rangle\big\}\Big\rangle \tag{4.14}$$

Thanks to the labels, the set of observations relative to *collectionConformance* can be filtered based in the data source and therefore be properly subject of controls relative to the data source in isolation.

4.4.3 Sessions and observations

Metrics, be them basic or derived, are generated in the context of a block session, be it a data, process, or compound block. Typically, basic metrics deliver one observation for each session they participate to, while derived metrics are calculated at the end of the session. Sub-process metrics may instead generate an arbitrary number of observations during a sub-process block session, to be the input of derived process metrics. To better understand, we consider two examples m-flows:

- Figure 4.10a shows an m-flow with a sequence of two processing blocks and a compound block enclosing them; the processing blocks have sensors:

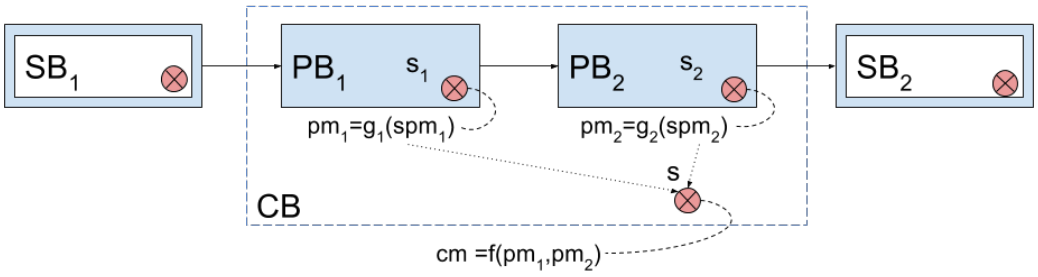
$$s_{1PB_1} = \big\langle \{spm_1\}, \{pm_1 = g_1(spm_1)\}, \emptyset, C_1 \big\rangle \tag{4.15}$$

$$s_{2PB_2} = \big\langle \{spm_2\}, \{pm_2 = g_2(spm_2)\}, \emptyset, C_2 \big\rangle \tag{4.16}$$

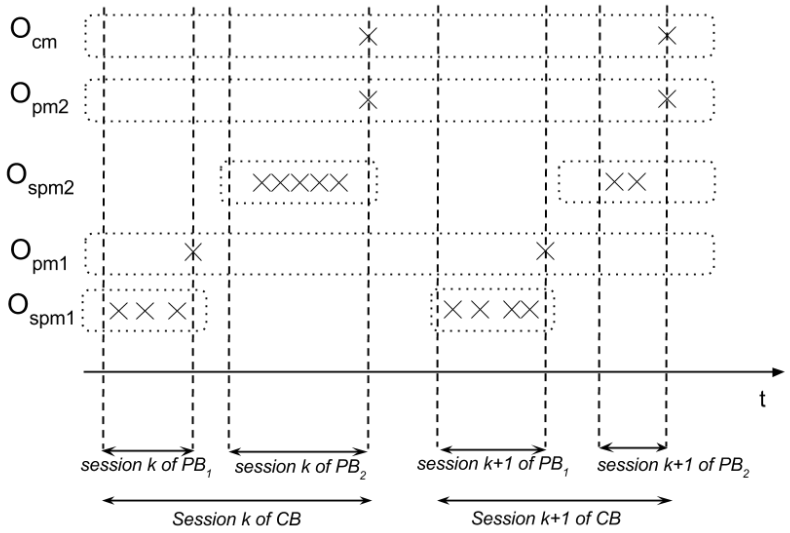
  while the compound block enclosing them has a sensor

$$s_{CB} = \big\langle \{pm_1, pm_2\}, \{cm = f(pm_1, pm_2)\}, C \big\rangle \tag{4.17}$$

  As shown by the time distribution of sessions, reported in Figure 4.10b, the observations relative to the compound metrics cm of CB are aligned with the closing time of the sessions relative to PB_2, which comes last in the ordering of monitoring. Controls, which are monitoring checks on specific time-predicate features of metrics observations, can be applied as follows: controls in C over O_{cm}, O_{pm_1}, and O_{pm_2}, while controls in C_1 to O_{pm_1} and controls in C_2 to O_{pm_2}. Observations relative to the basic sub-process metrics are instead "scattered" across different sessions, serve the generation of derived (process) metrics, and cannot be fed as input to the control engine.



(a)



(b)

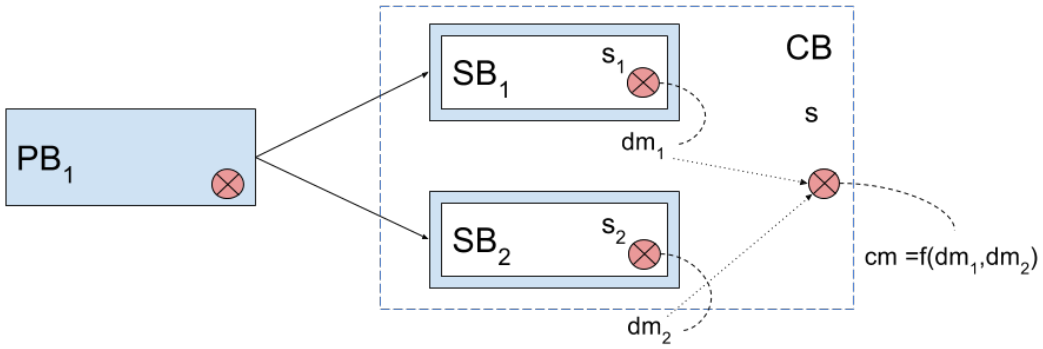
Figure 4.10: *M-flow with sequence processing blocks in a compound block and derived metrics*

- Figure 4.11a shows an m-flow with two storage blocks wrapped into a compound block. The storage blocks do not have dependencies, meaning their block sessions are not necessarily sorted, i.e. their observation are not expected to reach MoniQ in a specific order, and have sensors producing different metrics:

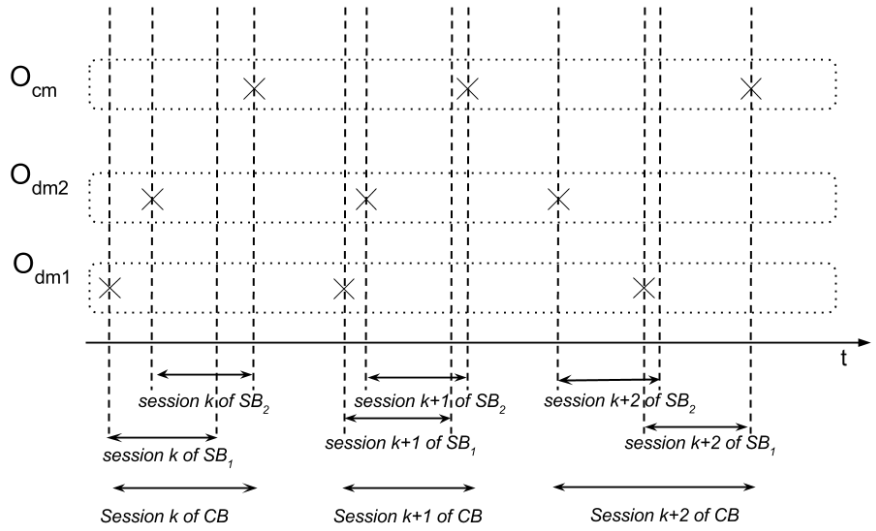
$$s_{1SB_1} = \langle \{dm_1\}, \emptyset, C_2 \rangle \tag{4.18}$$

$$s_{2SB_2} = \langle \{dm_2\}, \emptyset, C_2 \rangle \tag{4.19}$$

  The compound block CB serves the need of monitoring SB_1 and SB_2 in the same scope and at the end of the same compound block session. As shown in Figure 4.11b, observations relative to the dm_i's may arrive in any order as long as their respect the m-flow semantics, which requires one measurement for each data metrics in a block session. The compound metrics cm is calculated whenever both session blocks for SB_1 and SB_2 have correctly terminated, again in any ordering. As we shall illustrate in the following sections, monitoring controls can be applied over all metrics involved and the relative history of observations O_{cm}, O_{dm_1}, and O_{dm_2}.



(a)



(b)

Figure 4.11: *M-flow with parallel processing blocks in a compound block and derived metrics*

4.4.4 Data-flow aware monitoring

As mentioned above, DI quality managers first create an MoniQ m-flow, then need to ensure on the DI side that: (i) metrics are calculated and sent following the ordering encoded by blocks and relationship in the m-flow, and that (ii) closing notifications are sent when individual processing blocks should interrupt their operation. On the other hand, for the MoniQ framework presented so far, there are classes of m-flows for which such assumptions are not easy to enforce. For example, consider the m-flow relative to the literature aggregation data flow, presented in Figure 4.9. As mentioned above "the data flow executes weekly the same processing steps, possibly in parallel, over around 700 data sources DS_i". The m-flow collects labeled observations similar to the one reported in Equation 4.14.

The problem with these observations comes when the aggregation data flow is run in parallel, for example over different data sources, and several m-flow sessions have started contemporary. The sessions relative to different m-flow sessions of the data flow may overlap (e.g. several sessions relative to $collectionBlock$, one for each data source involved), hence metrics and closing notifications sent to MoniQ by such independent m-flows sessions may conflict and create inconsistencies. Parallelism is solved by introducing the notion of *data flow identifier*, *data flow-aware observation*, and *data flow-aware sessions*. As we shall see, the introduction of data flow-awareness propagates across all concepts of the framework.

Definition 4.9. A *data flow identifier* DFI is a unique reference generated by the DI to identify a set of measurements, i.e. observations, relative to the same data flow run. □

The implementation of metrics and closing notifications must be data flow-aware, that is using the same DFI to mark the messages they send to MoniQ are relative to the same data flow execution. DFI's are DI and application dependent. For example, in the literature aggregation example the DI quality manager could use the data source unique reference provided by the data infrastructure, under the assumption that the data flow will never run in parallel over the same data source. In all the cases where the DI data flows are implemented using workflow engines, DFI could directly be the identifier of the workflow instance spawned by the engine, which always exist in these scenarios.

Definition 4.10. A *data flow-aware observation* (*df-observation*) is an observation described by a tuple of the form:

$$o = \langle t, m, v, s_B, map \rangle_{DFI} \tag{4.20}$$

□

To cope with this notion MoniQ supports the notion of *data flow-aware sessions*. The introduction of df-sessions has several implications in the semantics of the sessions.

Definition 4.11. An *data flow-aware session* (*df-session*) is a session ss_{DFI} characterized by a specific DFI:

- *Storage block df-sessions*:
  - A storage block df-session ss_{DFI} starts when the first df-observation o_{DFI} relative to a data metrics of one of its sensors reaches MoniQ;

  - A storage block df-session ends correctly when all metrics of its sensors have sent df-observations o_{DFI};
  - The block session terminates incorrectly when a df-observation o_{DFI} relative to the metrics of the block is received twice before the correct termination of the session;
  - When the block session terminates, all compound metrics observations are calculated by MoniQ and tagged with DFI, obtained from the last and session closing observation received;

- *Processing block df-sessions*:
  - A processing block df-session ss_{DFI} starts when the first df-observation o_{DFI} relative to a sub-process metrics reaches MoniQ;
  - A processing block df-session ends correctly with a closing notification specifying block and data flow identifier DFI;
  - If a closing notification for block session ss_{DFI} is not sent, the session may remain "idle"; the MoniQ framework allows to set time-to-wait thresholds, after which the session is terminated incorrectly;
  - When the block session terminates, all process metrics and compound metrics of the sensors are calculated by MoniQ and tagged with DFI, obtained from the closing session notification;

- *Compound block df-sessions*:
  - A compound block df-session ss_{DFI} starts when the first df-session relative to DFI of one of the blocks it wraps starts;
  - A compound block df-session terminates correctly if the df-sessions relative to the blocks it wraps and tagged as DFI correctly terminate;
  - A compound block df-session incorrectly terminates if any of the df-sessions relative to the blocks it wraps incorrectly terminates;
  - When the block session terminates, all process and compound metrics observations are calculated by MoniQ and tagged with DFI, obtained from last sessions closed.

□

The semantics above, if supported by proper implementation of metrics and closing notifications, ensures a smooth, cascade-structured, monitoring of multiple parallel sessions relative to different executions of the same data flow. With respect to our requirements of generality, where we assumed that DI components and data flows may be of any kind, involving machine-driven workflow executions as well as humans executing different data processing phases by performing manual executions of scripts or applications, MoniQ can guarantee consistent monitoring of parallel data flow executions only under the conditions that the agents executing the workflows can provide an identity for the execution and make all metrics implementations and processing block closing notifications aware of it.

4.4.5 Controls

Having defined m-flows, relative sensors, and metrics, the last step for the DI quality manager is to configure the *controls* to be enforced by MoniQ. Controls are checks to be run over metrics observations in the context of a given sensor and relative block. Controls are performed when the session of the including block is terminated, that is when an observation for each of the sensor metrics is available, and can refer only to the metrics relative to their scope, identified by the sensor. Since controls are in principle applicable to all observations relative to all metrics of a sensor, in order to define controls we need first to introduce the notion of *sensor observation set*, then define *metrics selectors* and *sensor selectors* which allow to select the subset of observations of interest, and finally *sensor analyzers*, responsible to run the check over the observations returned by the selectors.

Definition 4.12. Let SS_s be the set of sessions relative to sensor s, and o_{ss,m_i} be the only observation generated for the metrics m_i in the session $ss \in SS_s$ (metrics produce one observation per session). Given a sensor s with a set of basic and derived metrics $M = \{m_i, \ldots, m_k\}$, we can define the *sensor observation set* as

$$O_s = \bigcup_{\forall ss \in SS_s} \left\{ \langle o_{ss,m_1}, \ldots, o_{ss,m_k} \rangle \right\} \tag{4.21}$$

O_s contains one "observation tuple" for each m-flow session so far terminated. □

O_s is the set of observation tuples obtained by sampling observations from all metrics from each session in the history of the sensor s (not to be confused with the concept of data provenance, which is the linage of data through the data flow). Figure 4.12 shows the last three tuples of O_s for the sensor s_{CB} in Figure 4.11a. Such a sampling somehow "puts in synch" all observations collected for s so far and allows the definition of controls which take into account the overall behaviour of the blocks SB_1 and SB_2 as modeled by the relative metrics.

Definition 4.13. Given a set (the history) of observations

$$O_m = \bigcup_{i=1}^{n} \left\{ \langle t_i, m, v_i, s, B, map \rangle \right\} \tag{4.22}$$

generated by a metrics m and the predicates p_v and p_l, a *metrics selector* $mSel$ returns a subset of O_m defined as:

$$mSel(O_m, p_v, p_l) = \left\{ o \in O_m | p_v(o.v) \wedge p_l(map) \right\} \tag{4.23}$$

where p_v is a predicate over the value v (e.g. less, equal or greater to a given value) and p_l defines a first order logic predicate over the set of attributes defined in map. □

Definition 4.14. A *sensor selector* is a component that allows to filter the tuples of a sensor observation set O_s in terms of time, in order to confine the tuples to:

- A given time range, i.e. select observation tuples such that their session closing time is between t_{start} and t_{end};

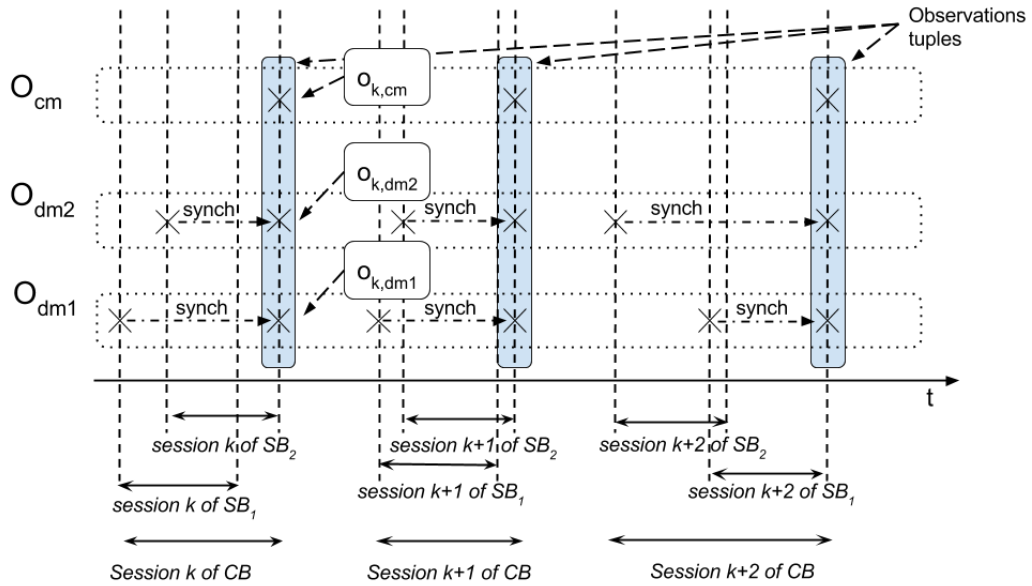


Figure 4.12: *Set of observations tuples*

- Number of sessions, i.e. select observation tuples from k sessions in the past (note that the number of sessions corresponds to the number of runs of underlying data flow).

and in terms of metrics selectors, one for each of the metrics involved:

$$sSel(O_s, p_t, mSel_{m_1}, \ldots, mSel_{m_k}) \tag{4.24}$$

The application of metrics selectors has a semantics that depends on the implementation, but in general we may assume that selectors may be applied according to a first order logic in order to make some of them mandatory for all observations in a tuple or at least one. □

Definition 4.15. Given a sensor s_B with a set of basic and derived metrics $M = \{m_i, \ldots, m_k\}$, an *analyzer* is a function $analyze$ that operates over a set of observation tuples O and returns a Boolean result $\{true/false\}$ if such a set meets given conditions, encoding the quality of block B. An analyzer can perform controls at different levels and not exclusively:

- Perform a control at the level of individual observations of one metrics m_i: a predicate p is applied to the observations o_i for all tuples in O, e.g. in order to ensure whether observations' values produced by m_i are below a certain threshold or not;
- Perform a control at the level of all observations of a metrics m_i: an aggregation function f is applied to the observations o_i for all tuples in O and then a predicate p is applied to the result, e.g. in order to ensure that the average of the observations' values produced by m_i is below a certain threshold;
- Perform a control at the level of individual tuples in O and involving a subset of metrics $M' \subseteq M$: a predicate p is applied to all tuples, which involves only the o_i's

relative to the metrics $m_i \in M'$, e.g. in order to ensure whether the observations' values produced by metrics m_1 are always below the observations' values produced by metrics m_2;

- Perform a control at the level of all tuples in O and involving a subset of metrics $M' \subseteq M$: an aggregation function f is applied to all tuples, which involves only the o_i's relative to the metrics $m_i \in M'$, then a predicate p is applied to the result, e.g. in order to check whether the average of observations' values produced by m_1 is below the average of observations' values produced by m_2.

An analyzer $analyze\Big(O, \big\{\langle f_1, p_1\rangle, \ldots, \langle f_m, p_m\rangle\big\}\Big)$ returns $true$ only if all its predicates p_i's return $true$. □

Finally, we can define a control as follows:

Definition 4.16. Given a sensor s with a set of basic and derived metrics $M = \{m_i, \ldots, m_k\}$, the set of observations O_s, a sensor selector $sSel(O_s, p_t, mSel_{m_1}, \ldots, mSel_{m_k})$, a *control* for s is defined as

$$c = analyze\Big(sSel\big(O_s, p_t, mSel_{m_1}, \ldots, mSel_{m_k}\big), \big\{\langle f_1, p_1\rangle, \ldots, \langle f_m, p_m\rangle\big\}\Big) \quad (4.25)$$

□

4.4.6 Actuators

The monitoring framework can provide feedback to the infrastructure thanks to an *actuator* component.

Definition 4.17. The *actuator* is a triggering mechanism that can be used for driving the DI data flow behavior in order to correct automatically or at least compensate one or more issues revealed by failing controls. The actuator is just an hook deployed within the DI which waits for "stimuli" from the monitoring framework in order to take specific countermeasures. For reasons concerning DI specificity, the business logic of such countermeasures must be provided by the user of the framework. □

The actuator can be leveraged in order to call dynamically a routine on the DI. In this case the actuator does not "reside" within the monitoring flow described in the monitoring scenario, but is deployed somewhere else in the DI data flow superintending, for example, management tasks or triggering other data flows. As an example, a data source could be automatically graylisted when the quality of the collected data drops down a certain level and whitelisted when the quality rises back to an expected value.

An actuator can also be used in order to perform dynamic *dynamic steering* of the data flow. Interesting works in this direction can be found in [66, 89, 91]. In this case the actuator resides within the data flow described by the monitoring and is used in order to select the most appropriate workflow path for execution. Consider for example the data flow reported in Figure 4.13. The DI consumes data from $data\ source$, prepares data by applying a pre-processing algorithm represented by $pre\ processing$, then applies one main algorithm out of $algorithm1$ and $algorithm2$, and finally stores results obtained in $data\ sink$. However, the designer of the data flow is aware that data provided by

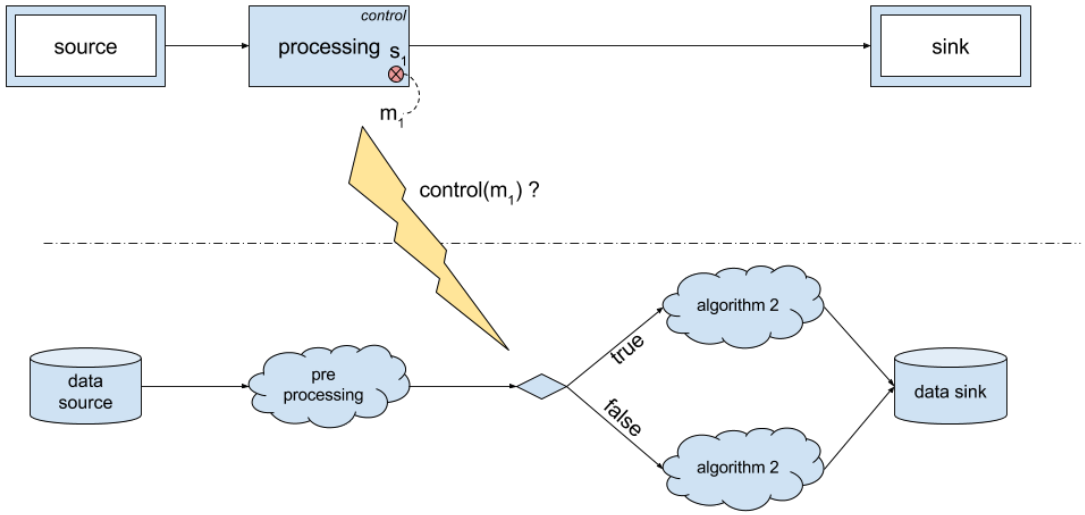


Figure 4.13: *An example showing how an actuator can be used in order to steer the data flow routing*

data source may carry unwanted features over time which can degrade the quality of the results produced by the algorithms; in particular, he is aware that one algorithm outperforms the other when data exhibit certain data flaws captured by metrics m_1; however this algorithm is much more CPU-intensive than the other and thus should be used only when strictly necessary. Hence, he takes advantage of MoniQ actuators in order to deploy a conditional data flow route which enables the choice of the two processing blocks *algorithm 1* and *algorithm 2* according to the control performed over metrics m_1.

As pointed out in [91], the presence of *dynamic steering*[3] and dynamic routine invocation introduced by actuators can make the data flow behavior non-deterministic and hence can introduce non-reproducibility of a particular data flow instance.

4.5 Integration effort required by MoniQ

The experienced effort in integrating MoniQ in the two DIs described in Section 5.1 and Section 5.2 is here reported by providing a rough estimation of the relevant time factors and impact of the distinct integration activities. Mainly, the integration effort of MoniQ is articulated over three phases: (i) *m-flow definition* to specify in which order, and of which type, the observations will flow from the monitored data flow to MoniQ; (ii) *monitoring intent definition* in order to specify which metrics, sensors and controls come into play within the given m-flow; and (iii) *data flow instrumentation* in order to put the metrics evaluations in place. For the three phases described, the relevant considerations in terms of complexity and integration effort are summarized as follows:

m-flow definition: the effort in terms of time spent for this phase is proportional to the complexity of the monitoring scenario. Since, as previously mentioned, there is no one-to-one correspondence between the DI data flow and the MoniQ monitoring flow,

[3] In literature the term "workflow navigation" can be found too

complex monitored data flows map, on average, to much simpler m-flows in MoniQ, as seen for example in our case studies. In the worst case scenario, a monitoring flow can be as complex as the relative data flow in terms of processing blocks and data storage blocks – i.e. one for each data flow component, either data or processing.

monitoring intent definition: for this phase stands the same evaluation discussed for the previous one. The more articulated is the monitoring intent, the bigger is the effort for defining sensors, metrics and controls over the specified m-flow.

data flow instrumentation: the complexity of this phase is intrinsic to the monitoring activity itself, as, if a certain observation needs to be extracted by a metrics and monitored, someone must provide the business logic for the extraction. However, the complexity of this task is independent from MoniQ and directly related to how easily the metrics can be implemented, and to the complexity of the monitoring intent as, in general, the more metrics need to be controlled, the more implementations must be provided. Regarding the complexity of metrics implementation, for example, the evaluation of the number of items pertaining to a given class of interest within a full-text index results into running a query over the index itself; while evaluating the same property for a raw dataset partitioned in multiple chunks stored in a distributed filesystem and not fitting into memory incurs a more "complex" implementation coping with typical "big data" issues. As another example, the evaluation of the completeness of a dataset element toward a fixed set of fields can be easily implemented (e.g. by running a batch of xQueries), while the evaluation of the compliance of the same record toward externally defined vocabularies/ontologies results in a slightly more complex implementation as access to external resources – i.e. the vocabulary/ontology definition – is involved. In Chapter 6, we will see how the proposed MoniQ-di-client simplifies further, in Java-based DI's data flows, the effort for the integration of the monitoring framework by factorising common aspects such as communication and metrics template evaluation.

CHAPTER 5

Experimentation and evaluation

In this chapter we validate the MoniQ framework proposed in Chapter 4 by going through the use cases presented in Chapter 2, namely the OpenAIRE and the CORE data infrastructures, and describing, for each one of them, the relative m-flows and monitoring intents using the proposed approach.

5.1 The OpenAIRE use case

MoniQ has been used to realize a monitoring system for the production environment of the OpenAIRE infrastructure. Taking as input the case study carried on in Section 2.1, in this section we outline the monitoring intents for our use cases – the aggregation, deduplication, inference and publishing data flows – using the formalism introduced in Chapter 4. For each one of them, we first briefly summarize the use case and then we describe how such monitoring intent can be realized with MoniQ by discussing the monitoring flow, the sensors, metrics and controls defined.

5.1.1 Monitoring the aggregation data flows

In this section we will cover the monitoring concerns outlined in Section 2.1.1. As anticipated, the OpenAIRE aggregation data flows deal with different types of data sources; the two being currently monitored are *literature data sources* and *entity registries*. The remainder of this section will follow the same structure.

Literature data sources The aggregation data flow operating with literature data sources collects XML records about publications exposed by data sources via an OAI-PMH endpoint, transforms the collected XML records, cleanses them, downloads the pointed PDFs and ultimately stores the resulting records, finally cleaned, into a document store

implemented with a MongoDB instance. The respective data flow is represented in Figure 5.1, which reports also the monitoring intent for the scenario.

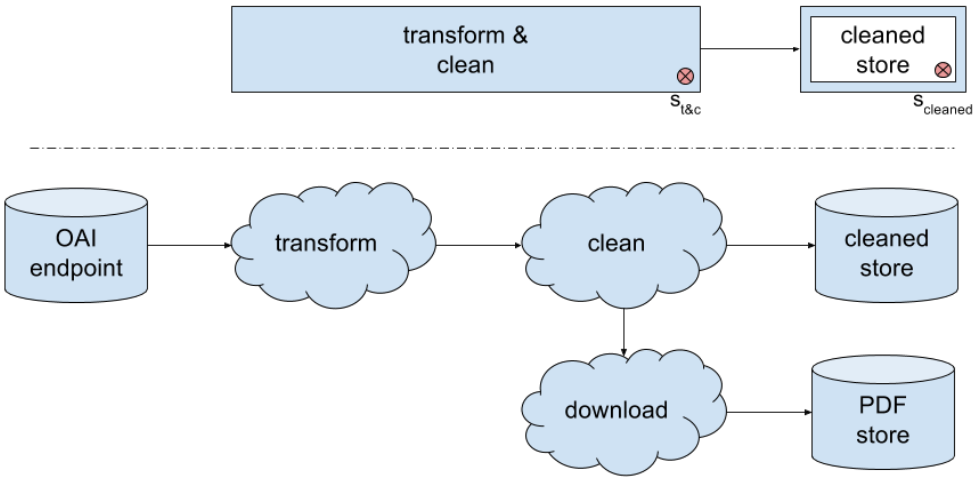


Figure 5.1: *Monitoring intents for the OpenAIRE literature aggregation data flow*

A first sensor $s_{t\&c}$ is defined in order to monitor the processing block $Transform\&clean$ and declares a sub-process metrics, called $sub\ OA\ compliance$, to be evaluated for each XML record (i.e. publication) processed in the data flow. Since the data flow can be instantiated for any of the data sources registered in OpenAIRE, we decided to leverage the observation's label mechanism when the Data Infrastructure (DI) data flow evaluates the metrics $sub\ OA\ compliance$ in order to mark observations produced by the respective sensor hook with a label `<sourceId, id>`. The metrics produces an observation stating how well an XML record "scores" in accordance to OpenAIRE guidelines[1], and it is implemented as a weighted composition of different scores associated to rules taking into account the presence/absence of certain fields (i.e. controlling if mandatory fields are non empty) and the corrected of values for vocabulary-controlled fields (i.e. controlling if the value respects the relative vocabulary; e.g. the *Access level* field[2]). It is important to ensure that at the end of the block session the process metrics $OA\ compliance$ – evaluated by averaging values of observations produced by $sub\ OA\ compliance$ for all records provided in the same aggregation round (i.e. in the same block session) – is at least 90% compliant to OpenAIRE guidelines, for any literature data source. Also, the trend of the average $OA\ compliance$ should be monotonic increasing for the last three sessions of aggregation, thus enforcing that the global perceived quality of the content provided by data sources is increasing over time.

Other two sub-process metrics are declared by sensor $s_{t\&c}$ in order to count, for every XML record, the number of fields the cleaning process successfully cleaned, $sub\ fields\ fixed$, and the number of fields that instead are skipped (i.e. when the cleaning process does not know how to reconcile the value), $sub\ fields\ skipped$. In this case, it is interesting to check whether the cleaning process yields (on average) more fixed fields than skipped ones in the cases when the average $OA\ compliance$ is less than 0.6; the

[1]The OpenAIRE guidelines, `http://guidelines.openaire.eu`

[2]The *Access level* field documentation can be found here: `https://guidelines.openaire.eu/en/latest/literature/field_accesslevel.html`. The field is controlled by the vocabulary described in `https://wiki.surfnet.nl/display/standards/info-eu-repo/#info-eu-repo-AccessRights`

contrary would mean the the cleaning rule is poorly configured and it is not effective. For this purpose, at the end of the block session the observations produced by the two sub-process metrics are averaged respectively into process metrics $sub\ fields\ fixed$ and $sub\ fields\ skipped$. The metrics and controls defined by sensor $s_{t\&c}$ are summarized in Table 5.1 and Table 5.2. In the tables presented hereafter, the following convention is adopted: the code *spm* indicate sub-process metrics, *pm* indicates process metrics, *dm* indicates data metrics and *cm* indicates compound metrics.

Table 5.1: *Metrics defined by sensor $s_{t\&c}$ over processing block $transform\&clean$*

Sensor $s_{t\&c}$			
Metrics	**Type**	**Description**	**Labels**
sub OA compliance	spm	compliance of an XML record about a publication towards OpenAIRE guidelines	sourceId=id
OA compliance	pm	$avg(sub\ OA\ compliance)$ for the block session	sourceId=id
sub fields fixed	spm	number of fields the cleaning process has corrected	sourceId=id
fields fixed	pm	$avg(sub\ fields\ fixed)$ for the block session	sourceId=id
sub fields skipped	spm	number of fields the cleaning process has skipped	sourceId=id
fields skipped	pm	$avg(sub\ fields\ skipped)$ for the block session	sourceId=id

Table 5.2: *Controls defined by sensor $s_{t\&c}$*

Sensor $s_{t\&c}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
OA compliance	last session	sourceId=*	Check whether the average OA compliance is above 90%
OA compliance	last 3 sessions	sourceId=*	Check if the average of OA compliance is monotonic increasing in the last 3 sessions
OA compliance fields fixed fields skipped	last session last session last session	sourceId=* sourceId=* sourceId=*	Check that when the average OA compliance is less than 0.6, the average number of fields fixed is greater than the average number of fields skipped

A second sensor, $s_{cleaned}$, is anchored to the storage block $cleaned\ store$ instead. The sensor declares the data metrics $total\ publications$ (see Table 5.3) which is evaluated by the DI data flow by simply running a query over the document collection persisted by MongoDb and returns the number of publications stored for a given data source. Again, the sensor hook (i.e. metrics implementation) shall take care of pushing the data source identifier characterizing the data flow execution into the labels of the generated observation. From a controls perspective, the metrics $total\ publications$, for every literature data source, should be monotonically increasing across subsequent data flow executions, thus enforcing that no content is actually being lost or accidentally no more provided. This control is reported in Table 5.4.

As can be noted, the usage of labels, used in these cases in order to track the identifier of the data source at hand, enables to reuse the same metrics name/implementation for every data source.

Table 5.3: *Metrics defined by sensor $s_{cleaned}$ over the storage block CleanedStore*

Sensor $s_{cleaned}$			
Metrics	**Type**	**Description**	**Labels**
total publications	dm	total number of publications	sourceId=id

Table 5.4: *Controls defined by sensor $s_{cleaned}$*

Sensor $s_{cleaned}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
total publications	last 3 sessions	sourceId=*	Check whether total publications is monotonic increasing

Entity registries The entity registry aggregation data flow collects records provided by an entity registry, transforms and cleans such records, and persists the aggregated results into a database realized with Postgres. The data flow is represented in Figure 5.2; the figure also shows the monitoring flow and the monitoring intent defined over the data flow of interest.

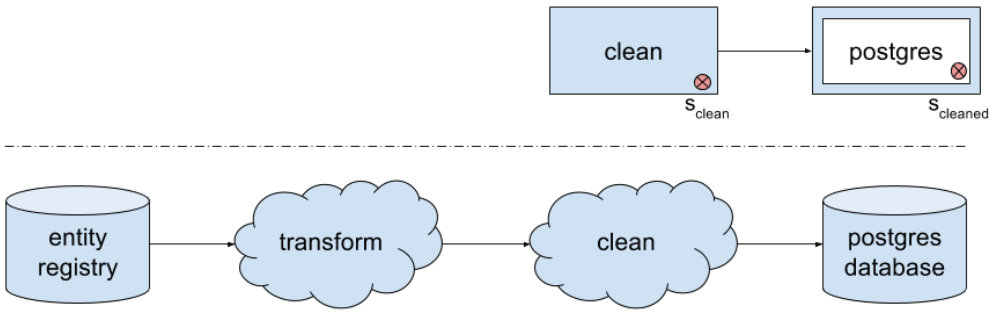


Figure 5.2: *Monitoring intents for the OpenAIRE entity registry aggregation data flow*

A first sensor s_{clean} is defined over processing block $clean$ and declares a sub-process metrics $sub\ compliance$ (to defined vocabularies) to be evaluated for each record exiting the cleaning process as follows:

$$sub\ compliance = \frac{N_{compliantFields}}{N_{vocabularyControlledFields}} \tag{5.1}$$

We expect that, when the block session of $clean$ is closed, the process metrics $compliance$ – evaluated by averaging the observations generated by metrics $sub\ compliance$ over the block session – is greater than 0.8 for the entity registry at hand. As seen already, observation's labels are used in order to discriminate among different entity registries. The metrics and controls for the sensor s_{clean} are summarized in Tables 5.5 and 5.6.

Moving forward, we defined a sensor $s_{postgres}$ attached to storage block $postgres$ and declaring two data metrics about the $completeness$ of two tables of interest in the database which are populated by the entity registry aggregation data flow: the $organizations$ and the $projects$ tables. An intuitive implementation for evaluating the

Table 5.5: *Metrics defined by sensor s_{clean} over the processing block clean*

Sensor s_{clean}			
Metrics	**Type**	**Description**	**Labels**
sub compliance	spm	compliance of an XML record toward defined controlled vocabularies	sourceId=id
compliance	pm	$avg(sub\ compliance)$ for the block session	sourceId=id

Table 5.6: *Controls defined by sensors s_{clean}*

Sensor s_{clean}			
	Selector(s)		**Analyzer**
Metrics	p_t	p_l	
compliance	last session	sourceId=*	the average compliance of XML should be greater than 0.8

completeness of a database table of N_c columns and N_t tuples could be, for example, to compute the average of each columns completeness, this being defined as the ratio of tuples having a $NULL$ value in that column over the number of tuples.

$$completeness_{table} = \frac{\sum_{i=1}^{N_c} completeness_{column_i}}{N_c} = \frac{\sum_{i=1}^{N_c} \frac{CountNull(column_i)}{N_t}}{N_c} \tag{5.2}$$

Other available (possibly more sophisticated) implementations for assessing completeness in relational databases can be found in literature [15]. In our case, we want to ensure that $organizations\ completeness$ and $projects\ completeness$ metrics are both monotonic increasing and above the 70% throughout the last three data flow executions. Since in this case we are not interested into monitoring the metrics on a per-entity-registry criteria, no label marking the observations produced is needed.

Another metrics declared by sensor $s_{postgres}$ takes into account the number of $misdated\ projects$ stored in the projects table of the database. The metrics implementation corresponds to a SQL query that counts the number of projects with impossible dating w.r.t. the relative funding stream; for example, FP7 projects declaring date ranges out of the range 2007-2015 are clearly a mistake. Such anomaly is expected to decrease monotonically over time as we hope that records will be cleaned and fixed right at the source by entity registries. The metrics and controls declared by the sensor $s_{postgres}$ are summarized in Tables 5.7 and 5.8.

Table 5.7: *Metrics defined by sensor $s_{postgres}$ over the storage block $postgres$*

Sensor $s_{postgres}$			
Metrics	**Type**	**Description**	**Labels**
organizations completeness	dm	degree of completeness of the table *organizations* persisted on Postgres	–
project completeness	dm	degree of completeness of the table *projects* persisted on Postgres	–
misdated projects	dm	number of projects with impossible date range w.r.t. the funding stream	–

Table 5.8: *Controls defined by sensors $s_{postgres}$*

Sensor $s_{postgres}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
organizations completeness	last 3 sessions	–	the completeness of organizations table must be monotonic increasing and in any case above 70%
projects completeness	last 3 sessions	–	the completeness of projects table must be monotonic increasing and in any case above 70%
misdated projects	last 3 sessions	–	the metrics is expected to be monotonic decreasing

5.1.2 Monitoring the deduplication data flow

In this section we will cover the monitoring concerns outlined in Section 2.1.2 for the deduplication data flow. The deduplication data flow reads all the cleaned content present in MongoDb (as produced from the literature aggregation data flow), stores it in a column store based on HBase and runs deduplication algorithms (more details about the deduplication system can be found in [5]) over two main entities of interest – *organizations* and *publications* – after a *clique* (i.e. a cluster of similar objects) has been discovered. The equivalence relationships among distinct objects that are found by the deduplication algorithms are then persisted in an actionset for later use. In particular, whenever a clique is discovered a representative record is elected and metadata information of clustered records is merged. The monitoring flow and the monitoring intent defined over it are depicted in Figure 5.3.

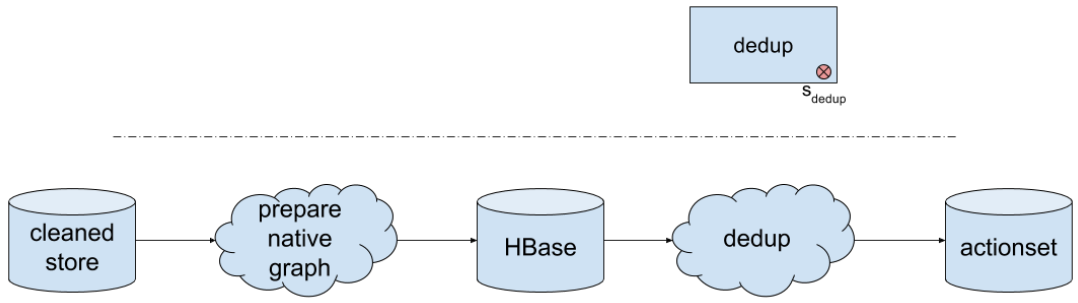


Figure 5.3: *Monitoring intents for the OpenAIRE deduplication data flow*

From a monitoring perspective, it is interesting to monitor the deduplication factor, i.e. the number of publications (organizations) that are merged into one representative publication (organization).

For this purpose, one sensor s_{dedup} is defined and anchored to the processing block $dedup$; the sensor declares one sub-process metrics, $sub\ dedup\ factor$, as reported in Table 5.9. The sensor receives an observation whenever the deduplication process generates a new representative record (either publications or organization) and tracks the number of records merged of both types. As can be seen, the usage of labels, in this case tracking whether the type of the generated representative record is a *publication* or an *organization*, permits to reuse the metrics' name and definition and adapt them to

both cases.

At the end of the block session, the process metrics $dedup\ factor$ is evaluated by averaging observations generated by metrics $sub\ dedup\ factor$ for organizations and publications, and then checked against the controls reported in Table 5.10 in order to ensure whether the average deduplication factors are in a bounded neighbourhood of the thresholds evaluated experimentally in [5].

Table 5.9: *Metrics defined by sensor s_{dedup} over the processing block $dedup$*

Sensor s_{dedup}			
Metrics	**Type**	**Description**	**Labels**
sub dedup factor	spm	number of publications (organizations) merged in a generated representative record	type={pubs | orgs}
dedup factor	pm	$avg(sub\ dedup\ factor)$ for the current session	type={pubs | orgs}

Table 5.10: *Controls defined by sensor s_{dedup}*

Sensor s_{dedup}			
	Selector(s)		**Analyzer**
Metrics	p_t	p_l	
dedup factor	last session	type=pubs	check if the average deduplication factor is within the $\pm 5\%$ variation of the experimental threshold set to 2.40
dedup factor	last session	type=orgs	check if the average deduplication factor is within the $\pm 5\%$ variation of the experimental threshold set to 2.14

5.1.3 Monitoring the inference data flow

In this section we will cover the monitoring concerns outlined in Section 2.1.3. The inference data flow processes PDFs collected by the aggregation subsystem (see Section 2.1.1) in the attempt of extracting full-texts of publications from them. Once the extraction process has terminated, the extracted full-texts and the deduplicated graph (produced by the provision subsystem) are taken as input for inference algorithms (more details about the inference subsystem can be found in [52]) that ultimately produces inferred relationships in an actionset. The inference data flow is represented in Figure 5.4, which shows also the monitoring flow and the monitoring intent defined on top of it.

As anticipated in Chapter 2, in this scenario we want to monitor some metrics extracted from the PDFs processed and from the inferred knowledge ultimately produced by the inference data flow. For this purpose two sensor are defined: $s_{extraction}$ and $s_{actionset}$. The sensor $s_{extraction}$ declares two dichotomous (i.e. binary) metrics evaluated for each PDF processed, namely the $sub\ pdfs\ corruption$ and $sub\ fulltext\ retrievability$, summarized in Table 5.11; the observations produced by the two sub-process metrics are then aggregated at the end of the block session into the process metrics $pdfs\ corruption$ and $fulltext\ retrievability$ respectively. On these two latter metrics, we want to check whether the number of valid PDFs and the number of full-texts actually retrievable are above a certain threshold in order to guarantee that the inference process is working under the correct assumptions. In particular, the number of invalid pdfs must be less

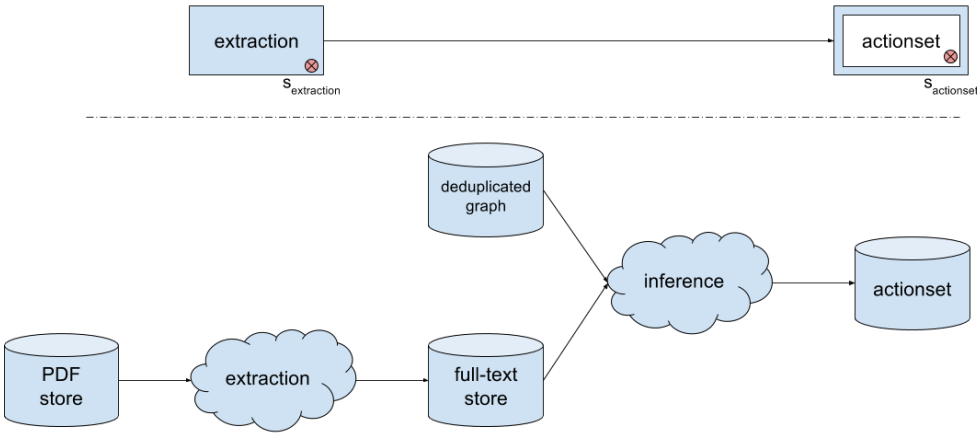


Figure 5.4: *Monitoring intents for the OpenAIRE inference data flow*

than 10% and monotonically decreasing throughout time, while a fulltext should be successfully extracted at least in the 90% of the cases. A summary of the controls defined is reported in Table 5.12.

Table 5.11: *Metrics defined by sensor $s_{extraction}$ on processing block extraction*

Sensor $s_{extraction}$			
Metrics	**Type**	**Description**	**Labels**
sub pdfs corruption	spm	generates 1 if the PDF is broken (i.e. it is not possible to open the file), 0 otherwise	–
pdfs corruption	pm	summation of *sub pdfs corruption* over the current session	–
sub fulltext retrievability	spm	generates 1 if a valid full-text can be extracted form the PDF, 0 otherwise	–
fulltext retrievability	pm	summation of *sub fulltext retrievability* over the current session	–

Table 5.12: *Controls defined by sensor $s_{extraction}$*

Sensor $s_{extraction}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
pdfs corruption	last session	–	the number of corrupted PDFs processed must be less than the 10% of the total number of PDFs processed
pdfs corruption	last 3 sessions	–	the number of corrupted PDFs processed should decrease monotonically over time
fulltext retrievability	last session	–	the number of valid full-texts extracted from valid PDFs must be above 90% of the total number of valid PDFs processed

Another sensor $s_{actionset}$ is instead anchored to the storage block $actionset$ in order to monitor the final product of the entire inference process. The sensor declares several

data metrics about the actionset produced by the last execution of inference data flow. These metrics mainly are about the numbers of different relationships that the inference process has generated, e.g. links from publications to projects, publication similarity, citations among publications, links among publications and datasets, affiliations between publications and organizations. A summary of the metrics declared by $s_{actionset}$ is reported in Table 5.13. Against such defined data metrics, controls should ensure a monotonically increasing trend with a max 5% percent variation among subsequent inference data flow executions (i.e. data flow sessions), as reported in Table 5.14.

Table 5.13: *Metrics defined by sensor $s_{actionset}$ on storage block $actionset$*

Sensor $s_{actionset}$			
Metrics	**Type**	**Description**	**Labels**
inferred pub2proj	dm	number of inferred relationships among publications and projects funding the research efforts	–
inferred simrels	dm	number of inferred similarity relationships among publications	–
inferred citations	dm	number of inferred citations among publications	–
inferred pub2data	dm	number of inferred relationships between publications and datasets	–
inferred affiliations	dm	number of inferred affiliation among publications and organizations	–

Table 5.14: *Controls defined by sensor $s_{actionset}$*

Sensor $s_{actionset}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
inferred pub2proj	last 2 sessions	–	the number of inferred relationships between publications and projects should be monotonic increasing in last two inference executions, but it should not increase more than 5%
inferred simrels	last 2 sessions	–	the number of inferred similarity relationships among publications should be monotonic increasing in last two inference executions, but it should not increase more than 5%
inferred citations	last 2 sessions	–	the number of inferred citations among publications should be monotonic increasing in last two inference executions, but it should not increase more than 5%
inferred pub2data	last 2 sessions	–	the number of inferred relationships between publications and datasets should be monotonic increasing in last two inference executions, but it should not increase more than 5%
inferred affiliations	last 2 sessions	–	the number of inferred affiliations between publications and organizations should be monotonic increasing in last two inference executions, but it should not increase more than 5%

5.1.4 Monitoring the publishing data flow

In this section we will cover the monitoring concerns previously outlined in Section 2.1.4. The publishing data flow takes care of the materialization of the "deduplicated and enriched information graph" produced by the provision subsystem (see Section 2.1) into four different data components serving separated use cases. The publishing data flow, as well as the monitoring intent defined over it, are reported in Figure 5.5.

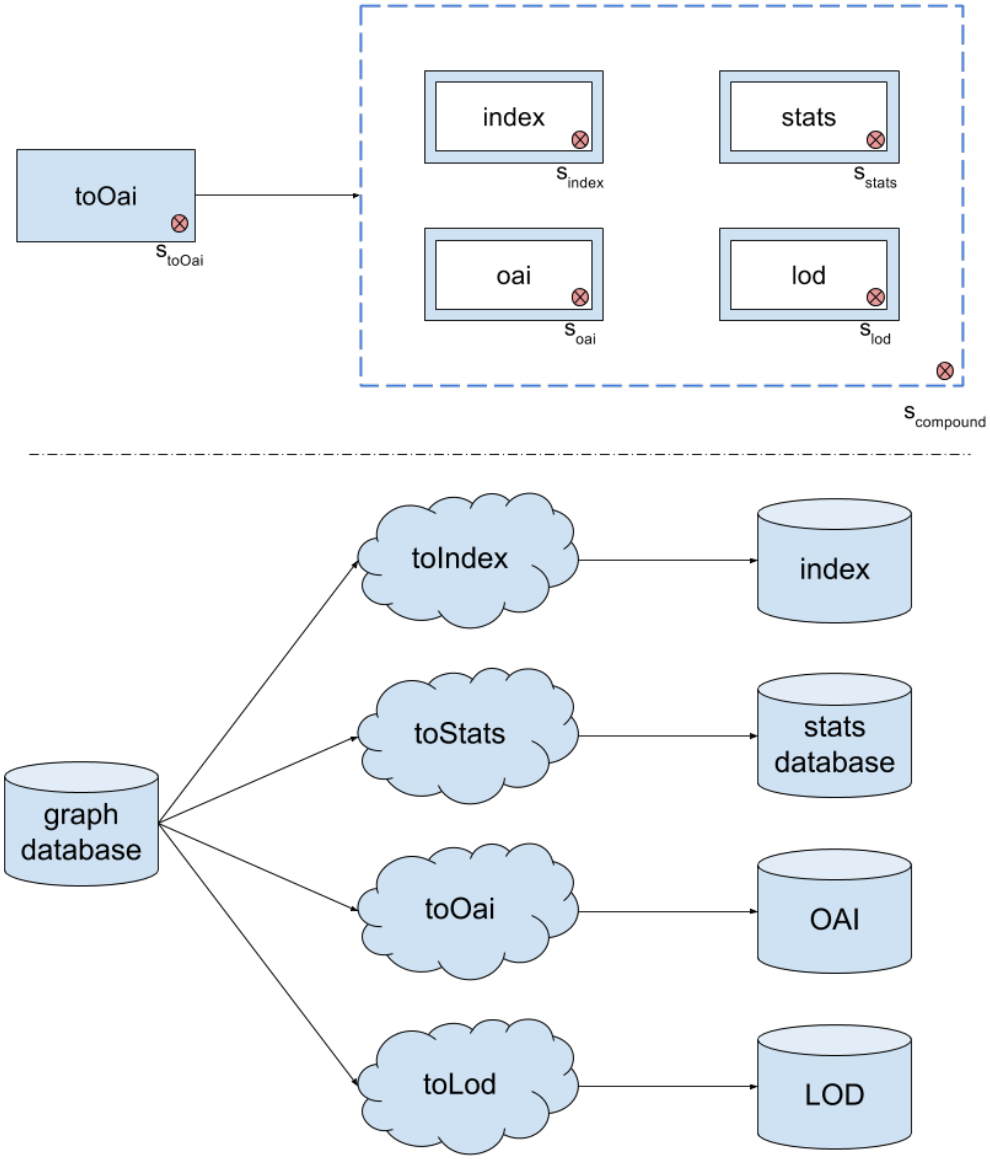


Figure 5.5: *Data flow and monitoring intents defined for the OpenAIRE publishing data flow scenario*

The only processing component being monitored is $toOai$, which takes care of transforming publications stored in the information graph into XML records compliant to Dublin Core to be exported via OpenAIRE's OAI-PMH endpoint[3]. The relative sensor, s_{toOai}, is implemented according to the specifications reported in Table 5.15. Whenever the data flow has a record ready to be stored into the OAI storage component, the sensor

[3]OpenAIRE API documentation, `http://api.openaire.eu/#cha_oai_pmh`

hook will assess the two declared sub-process metrics over the XML record and then send the relative observations to MoniQ. The implementation of metrics $sub\ completeness$ runs a batch of $n_{desiredFields}$ xQueries over a set of xpaths defined by the user, counts the number of non-empty fields found $n_{initializedFields}$ and generates an observation containing the ratio

$$sub\ completeness = \frac{N_{initializedFields}}{N_{desiredFields}} \tag{5.3}$$

Regarding the metrics $sub\ compliance$, its implementation checks $n_{vocabularyControlledFields}$ fields via xpaths and checks whether the content of each field is compliant to the vocabularies intended to control the fields. It finally emits the ratio

$$sub\ compliance = \frac{N_{compliantFields}}{N_{vocabularyControlledFields}} \tag{5.4}$$

where $n_{compliantFields}$ is the number of fields whose content conforms to what is specified by the relevant controlled vocabulary. As can be noted, the metrics implementation in this case needs external resources (i.e. the vocabularies), hence it is necessary to load them into memory in order to access them during metrics evaluation; in our current implementation, the metrics implementation retrieves the vocabularies at runtime from the D-NET Registry [57]. Also, notice that these two metrics remind of the metrics $sub\ OA\ compliance$ introduced in Section 5.1.1; in this case the same qualitative indication about records is extracted by "promoting" the two factors, i.e. field completeness and vocabulary compliance, to metrics on their own, while previously were implicitly "hidden" into the implementation of the metrics $sub\ OA\ compliance$. Since we desire to monitor these two metrics for a specific group of OAI sets out of the many exposed by the OpenAIRE OAI-PMH endpoint, we decided to leverage observation's labels in order to store along observations the contextual information about the OAI sets the measurement is about. The OAI sets of interest for this use case are the following:

- *FP7Publications*: an OAI set containing all FP7 publications;
- *H2020Publications*: an OAI set containing all H2020 publications;
- *Open Access*: an OAI set containing all Open Access (OA) publications;
- *per-data-source sets*: a dedicated OAI set is created in OpenAIRE for each data source. Each set exposes publications belonging to one data source only.

When the block session has ended, observations produced by the two metrics are averaged into the process metrics $completeness$ and $compliance$ respectively. Over these two declared process metrics, s_{toOai} defines the following controls. As for $completeness$, the average completeness of records should be above 70% for each one of the sets listed above; while regarding the metrics $compliance$, we want to ensure that the average compliance of records prepared to be exported via OAI-PMH is above 90% for each OAI set. Summarizing, the controls defined over s_{toOai} are reported in Table 5.16.

The other sensors pictured in the monitoring scenario are collection sensors, one for each final storage block of the monitoring flow. The first two data components targeted for the integration of the monitoring function have been the full-text index and the database for statistics. The monitoring intents over OpenAIRE OAI and LOD

Table 5.15: *Metrics defined by sensor s_{toOai} over processing block $toOai$*

Sensor s_{toOai}			
Metrics	**Type**	**Description**	**Labels**
sub completeness	spm	the percentage of initialized (non empty) fields in an XML record advertised by OpenAIRE via OAI-PMH w.r.t a set of desired ones	set={FP7 ∣ H2020 ∣ OA ∣ *sourceId*}
completeness	pm	$avg(sub\ completeness)$ for the current session	set={FP7 ∣ H2020 ∣ OA ∣ *sourceId*}
sub compliance	spm	the percentage of vocabulary-controlled fields in an XML record advertised by OpenAIRE via OAI-PMH which are compliant to the relevant vocabulary	set={FP7 ∣ H2020 ∣ OA ∣ *sourceId*}
compliance	pm	$avg(sub\ compliance)$ for the current session	set={FP7 ∣ H2020 ∣ OA ∣ *sourceId*}

Table 5.16: *Controls defined by sensor s_{toOai}*

Sensor s_{toOai}			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
completeness	last session	set=*	Check whether the average completeness of records exported via OAI-PMH is above 70%
compliance	last session	set=*	Check whether the average compliance of records exported via OAI-PMH towards vocabularies is above 90%

endpoints are currently under integration; nonetheless, monitoring requirements are clearly outlined.

The sensor s_{index} declares data metrics (for the sake of space, a subset of the metrics declared is reported in Table 5.17) over the collection of records indexed by Solr backend; each metrics is implemented by running a query expressed with Solr syntax that essentially returns a count over the records matching the clauses expressed. As an example, the query `deletedbyinference:false AND resulttypeid:publication` retrieves the total number of publications indexed by OpenAIRE. The controls defined over these metrics are reported in Table 5.18. As can be noted, the controls defined by s_{index} either check that any of the metrics generated is monotonic increasing or monotonic decreasing over time (i.e. across last n data flow sessions).

Similarly, s_{stats} declares data metrics about statistics computed and advertised by OpenAIRE. The implementation of these metrics simply runs a query by key over a key-value collection persisted on Redis and returns the associated value (i.e a single stats number). For the sake of space, a selected subset of the metrics defined by the sensor s_{stats} are reported in Table 5.19. As we will point out later in this section, some metrics declared by s_{stats} are mainly intended for inter-backend alignment purposes (see sensor $s_{compound}$ defined later). Over the metrics peculiar to s_{stats} (i.e. statistics about interlinking results) we want to ensure that the relative trends are monotonic increasing, as shown in Table 5.20.

The sensor s_{lod}, whose data metrics are reported in Table 5.21, declares metrics that need to be evaluated over the LOD graph stored in Virtuoso by querying the data component (i.e. SPARQL queries). At the time of writing, we are in particular interested in metrics related to the main entities *projects*, *datasets*, *publications* and their monotonic increasing trend, as reflected by the controls defined over them reported in Table 5.22. In the future, other meaningful metrics will be added along these ones in order to assess and track the "goodness" of the reasoning and interlinking algorithms producing and enriching the LOD graph throughout time.

The s_{oai} declares data metrics to be evaluated against the collections of documents exported by OAI-PMH. The metrics exported are defined as reported in Table 5.23. In particular, each metrics returns the total number of publications exported by the OAI sets already mentioned above in this section. Over such metrics, the controls reported in Table 5.24 are defined, mainly checking their trends for increasing monotonicity.

As can be seen, several of the metrics declared by these last four sensors deal with different funders and funding streams, namely: FP7, H2020, Wellcome Trust[4] (WT), the Fundação para a Ciência e a Tecnologia[5] (FCT), the Australian Research Council[6] (ARC) and the Australian National Health and Medical Research Council[7] (NHMRC). This is done in order to provide accountability to funders w.r.t the content provided to the general public.

As anticipated, the four different materialization processes $toIndex$, $toOai$, $toStats$, $toLod$ are supposedly going to finish at distinct times; however, we want to ensure that the distinct data components they populate are mutually aligned at the end of the data flow (i.e. consistency, and thus quality, of the data components and of the processes feeding data into them). For this reason, we defined a compound block enclosing the four storage blocks $index$, $stats$, oai and lod, and thus inheriting the scope of their sensors s_{index}, s_{stats}, s_{lod} and s_{oai}. This enables us to define the compound sensor $s_{compound}$ declaring the compound metrics $misleading\ FP7\ pubs$, reported in Table 5.25, accounting for the number of publications whose publication date in prior to FP7 time extension (i.e. 2007-2015), but whose link-to-project points to a project funded by FP7. The controls to be ensured in this monitoring scenario are defined on $s_{compound}$ and are reported in Table 5.26 and mainly deal with the alignment of the four data components, while the compound metrics $misleading\ FP7\ pubs$ is checked for a monotonic decreasing trend throughout time.

[4]Wellcom Trust, `https://wellcome.ac.uk/funding`
[5]FCT, `https://www.fct.pt/apoios`
[6]ARC, `http://www.arc.gov.au/grants`
[7]NHMRC, `https://www.nhmrc.gov.au/grants-funding`

Table 5.17: *Metrics defined by sensor s_{index} over the storage block $index$*

Sensor s_{index}			
Metrics	**Type**	**Description**	**Labels**
projects	dm	number of projects	block=index
results	dm	number of research outputs (datasets or publications)	block=index
datasets	dm	number of datasets	block=index
publications	dm	number of publications	block=index
OA publications	dm	number of open access publication	block=index
closed access pubs	dm	number of publications with closed access restriction	block=index
FP7 publications	dm	number of publications funded by FP7 projects	block=index
FP7 OA publications	dm	number of OA publications funded by FP7 projects	block=index
FP7 projects	dm	number of projects belonging to FP7	block=index
FP7 projects SC39	dm	number of projects belonging to FP7 and declaring Special Clause 39	block=index
FP7 ongoing projects	dm	number of FP7 projects which are still ongoing	block=index
FP7 proj no end date	dm	number of FP7 projects whose end date is unknown	block=index
FP7 pubs in 2007-2015	dm	number of FP7 publications funded by FP7 projects, whose publication date (correctly) falls within FP7 time extension	block=index
old pubs w\FP7 projects	dm	number of publications whose publication date is prior to FP7 time extension, but that are claimed to be funded by an FP7 project	block=index
H2020 publications	dm	number of publications funded by H2020 projects	block=index
H2020 OA publications	dm	number of OA publications funded by H2020 projects	block=index
H2020 projects	dm	number of projects funded by H2020	block=index
H2020 ongoing projects	dm	number of H2020 projects that are still ongoing	block=index
H2020 proj no end date	dm	number of H2020 projects whose end date is unknown	block=index
old pubs w\H2020 proj	dm	number of publications whose publication date is prior to H2020 time extension, but that are claimed to be funded by an H2020 project	block=index
WT projects	dm	number of Wellcome Trust (WT) projects	block=index
WT publications	dm	number of publications funded by WT projects	block=index
WT open access pubs	dm	number of open access publications funded by WT	block=index
ERC projects	dm	number of projects funded by ERC	block=index
ERC publications	dm	number of publications funded by ERC projects	block=index
ERC open access pubs	dm	number of open access publications funded by ERC	block=index
EGI publications	dm	number of publications funded by EGI projects	block=index
EGI open access pubs	dm	number of open access publications funded by EGI	block=index
FET publications	dm	number of publications funded by FET projects	block=index
FET open access pubs	dm	number of open access publications funded by FET	block=index
FCT projects	dm	number of projects funded by FCT	block=index
FCT publications	dm	number of publications funded by FCT projects	block=index
FCT open access pubs	dm	number of open access publications funded by FCT	block=index
ARC projects	dm	number of projects funded by ARC	block=index
ARC publications	dm	number of publications funded by ARC	block=index
NHMRC projects	dm	number of projects funded by NHMRC	block=index
NHMRC publications	dm	number of publications funded by NHMRC	block=index
missing titles	dm	number of publications whose title is unknown	block=index

Table 5.18: *Controls defined by sensor* s_{index}

Sensor s_{index}			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
projects	last 3 sessions	block=index	the number of projects must be increasing over time
results	last 3 sessions	block=index	the number of research products must be increasing over time
datasets	last 3 sessions	block=index	the number of datasets must be monotonic increasing over time
publications	last 3 sessions	block=index	the number of publications must be monotonic increasing over time
OA publications	last 3 sessions	block=index	the number of OA publications must be increasing over time
FP7 projects	last 3 sessions	block=index	the number of FP7 projects should be monotonic increasing over time
FP7 publications	last 3 sessions	block=index	the number of publications funded by FP7 should be monotonic increasing over time
FP7 OA publications	last 3 sessions	block=index	the number of OA publications funded by FP7 should be monotonic increasing over time
FP7 ongoing projects	last 3 sessions	block=index	the number of ongoing FP7 project should decrease over time
FP7 proj no end date	last 3 sessions	block=index	the number of FP7 projects with unknown end date should decrease monotonically over time
old pubs w\FP7 projects	last 3 sessions	block=index	the number of these publication should decrease over time
H2020 projects	last 3 sessions	block=index	H2020 projects should increase throughout time
H2020 publications	last 3 sessions	block=index	H2020 publication should increase over time
H2020 OA publications	last 3 sessions	block=index	the number of H2020 OA publication should monotonically increase over time
H2020 ongoing projects	last 3 sessions	block=index	the number of onjoing H2020 project should increase over time
H2020 proj no end date	last 3 sessions	block=index	the number of project with missing end date should decrease over time
old pubs w\H2020 projects	last 3 sessions	block=index	the number of these publication should monotonically decrease over time
...	...	...	...
missing titles	last 3 sessions	block=index	the number of publications with missing title should be monotonic decreasing

Table 5.19: *Metrics defined by sensor s_{stats} over the storage block stats*

Sensor s_{stats}			
Metrics	**Type**	**Description**	**Labels**
projects	dm	number of projects	block=stats
results	dm	number of research outputs (datasets or publications)	block=stats
datasets	dm	number of datasets	block=stats
publications	dm	number of publications	block=stats
OA publications	dm	number of open access publications	block=stats
closed access pubs	dm	number of closed access publications	block=stats
FP7 publications	dm	number of publications funded by FP7	block=stats
FP7 OA publications	dm	number of OA publications funded by FP7	block=stats
FP7 projects w\pubs	dm	number of projects funded by FP7 with linked publications	block=stats
FP7 projects	dm	number of projects funded by FP7	block=stats
FP7 projects SC39	dm	number of projects funded by FP7 with Special Clause 39 (SC39)	block=stats
FP7 projects SC39 w\pubs	dm	number of projects funded by FP7 with SC39 with linked publications	block=stats
H2020 publications	dm	number of publications funded by H2020	block=stats
H2020 OA publications	dm	number of OA publications funded by H2020	block=stats
H2020 projects	dm	number of projects funded by H2020	block=stats
H2020 projects w\pubs	dm	number of projects funded by H2020 with linked publications	block=stats
WT projects	dm	number of projects funded by Wellcome Trust	block=stats
WT projects w\pubs	dm	number of projects funded by Wellcome Trust with linked publications	block=stats
WT publications	dm	number of publications funded by Wellcome Trust	block=stats
WT open access pubs	dm	number of open access publications funded by Wellcome Trust	block=stats
ERC projects	dm	number of projects funded by ERC	block=stats
ERC projects w\pubs	dm	number of projects funded by ERC with linked publications	block=stats
ERC open access pubs	dm	number of OA publication funded by ERC	block=stats
ERC publications	dm	number of publication funded by ERC	block=stats
datasources w\pubs	dm	number of datasource with linked publication	block=stats
EGI projects	dm	number of projects funded by EGI	block=stats
EGI projects w\pubs	dm	number of projects funded by EGI with linked publications	block=stats
EGI publications	dm	number of publications funded by EGI	block=stats
EGI open access pubs	dm	number of OA publications funded by EGI	block=stats
FCT publications	dm	number of publications funded by FCT	block=stats
FCT projects	dm	number of projects funded by FCT	block=stats
FCT projects w\pubs	dm	number of projects funded by FCT with linked publications	block=stats
FCT open access pubs	dm	number of open access publications funded by FCT	block=stats
FET publications	dm	number of publications funded by FET	block=stats
FET open access pubs	dm	number of open access publications funded by FET	block=stats
datasets linked to projects	dm	number of datasets linked to projects	block=stats
pubs linked to datasets	dm	number of publications linked to datasets	block=stats

Table 5.20: *Controls defined by sensor s_{stats}*

Sensor s_{stats}			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
datasets linked to projects	last 3 sessions	block=stats	the number of datasets linked to projects should be monotonically increasing
pubs linked to datasets	last 3 sessions	block=stats	the number of publications linked to datasets should be monotonically increasing
datasources w\pubs	last 3 sessions	block=stats	the number of data sources with publications should be monotonically increasing
FP7 projects SC39 w\pubs	last 3 sessions	block=stats	the number of projects funded by FP7 with special clause 39 should be monotonic increasing
EGI projects w\pubs	last 3 sessions	block=stats	the number of projects funded by EGI should be monotonic increasing
ERC projects w\pubs	last 3 sessions	block=stats	the number of projects funded by ERC should be monotonic increasing
FCT projects w\pubs	last 3 sessions	block=stats	the number of projects funded by FCT should be monotonic increasing
WT projects w\pubs	last 3 sessions	block=stats	the number of projects funded by WT should be monotonic increasing

Table 5.21: *Metrics defined by sensor s_{lod} over the storage block LOD*

Sensor s_{lod}			
Metrics	**Type**	**Description**	**Labels**
projects	dm	the number of projects exported as LOD graph	block=lod
datasets	dm	the number of datasets exported as LOD graph	block=lod
publications	dm	the number of publication exported as LOD graph	block=lod

Table 5.22: *Controls defined by sensor s_{lod}*

Sensor s_{lod}			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
projects	last 2 sessions	block=lod	the number of projects should be monotonic increasing
datasets	last 2 sessions	block=lod	the number of datasets should be monotonic increasing
publications	last 2 sessions	block=lod	the number of publications should be monotonic increasing

Table 5.23: *Metrics defined by sensor s_{oai} over storage block OAI*

Sensor s_{oai}			
Metrics	**Type**	**Description**	**Labels**
FP7 publications	dm	total number of publications funded by FP7	block=oai
H2020 publications	dm	total number of publications funded by H2020	block=oai
OA publications	dm	total number of OA publications	block=oai
publications	dm	total number of publications	block=oai set=id

Table 5.24: *Controls defined by sensor s_{oai}*

Sensor s_{oai}			
	Selector(s)		**Analyzer**
Metrics	p_t	p_l	
FP7 publications	last 2 sessions	block=oai	the number of FP7 funded publication should be monotonic increasing
H2020 publications	last 2 sessions	block=oai	the number of H2020 funded publication should be monotonic increasing
OA publications	last 2 sessions	block=oai	the number of open access publication should be monotonic increasing
publications	last 2 sessions	block=oai; set=*	the number of publication (per repository) should be monotonically increasing

Table 5.25: *Metrics defined by sensor $s_{compound}$*

Sensor $s_{compound}$			
Metrics	**Type**	**Description**	**Labels**
misleading FP7 pubs	cm	number of publications whose publication date in prior to FP7 time extension (2007-2015), but whose link-to-project points to a project funded by FP7. Evaluated as: $FP7\ publications - FP7\ publications\ in\ 2007\ 2015$	–

Table 5.26: *Controls defined by* $s_{compound}$

Sensor $s_{compound}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
misleading FP7 pubs	last session	–	check that the number of misleading FP7 publications is monotonic decreasing
projects	last session	block=index	check that the three numbers are aligned
projects	last session	block=stats	
projects	last session	block=lod	
datasets	last session	block=index	check that the three numbers are aligned
datasets	last session	block=stats	
datasets	last session	block=lod	
publications	last session	block=index	check that the three numbers are aligned
publications	last session	block=stats	
publications	last session	block=lod	
OA publications	last session	block=index	check that the three numbers are aligned
OA publications	last session	block=stats	
OA publications	last session	block=oai	
FP7 publications	last session	block=index	check that the three numbers are aligned
FP7 publications	last session	block=stats	
FP7 publications	last session	block=oai	
FP7 OA publications	last session	block=index	check that the two numbers are aligned
FP7 OA publications	last session	block=stats	
H2020 publications	last session	block=index	check that the three numbers are aligned
H2020 publications	last session	block=stats	
H2020 publications	last session	block=oai	
H2020 OA publications	last session	block=index	check that the two numbers are aligned
H2020 OA publications	last session	block=stats	
WT publications	last session	block=index	check that the two numbers are aligned
WT publications	last session	block=stats	
ERC publications	last session	block=index	check that the two numbers are aligned
ERC publications	last session	block=stats	

5.2 The CORE use case

MoniQ has been used to realize an experimental case study for monitoring the production environment of the CORE infrastructure. Taking as input the preliminary analysis exposed in Section 2.2, in this section we outline the monitoring intents for the case study using the formalism introduced in Chapter 4.

CORE data flow has been reported in Figure 2.9 in terms of processing component and data components; to the data flow corresponds the monitoring flow and the monitoring intents represented in Figure 5.6. As a recap, for each repository registered in the CORE

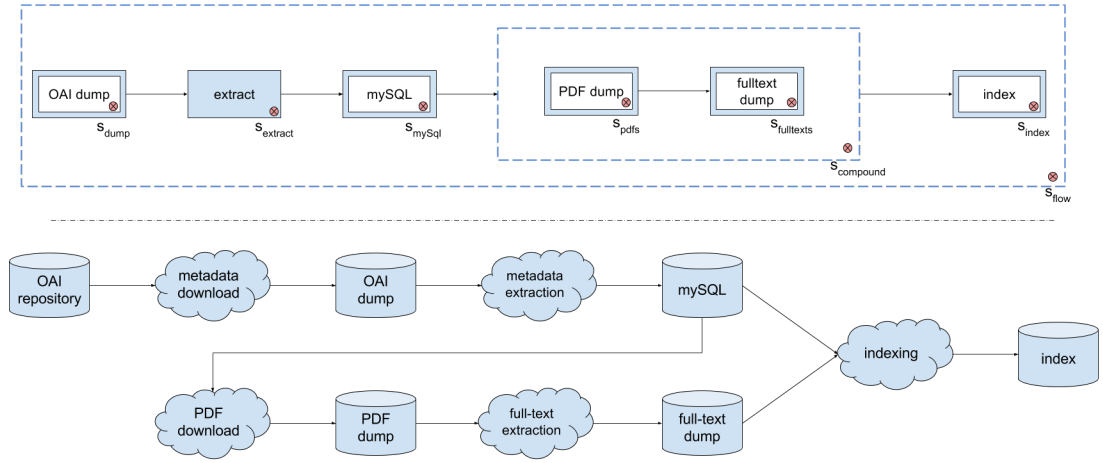


Figure 5.6: *Monitoring intent for CORE scenario*

infrastructure, the system enacts the data flow which provides to (i) download metadata from the repository of interest and extract information from the metadata retrieved, (ii) download PDFs of publications advertised by the repository and try to extract the full-text from each PDF retrieved, and finally (iii) index everything in a full-text index service user queries coming from the CORE web portal. Such data flow has to be monitored in several different ways in order to ensure that the results are consistent throughout its entirety and over time.

A first sensor s_{dump} is defined over the storage block $OAI\ dump$ and declares a metrics, called $total\ articles$, which evaluates the number of articles contained in the OAI dump file stored on the filesystem by the data flow. The number is extracted by running an xQuery counting the number of `<dc:record>` elements contained in the dump. Such number is expected to be monotonic increasing over time, as a negative variation would indicate that an issue has occurred either at the repository (e.g. records missing or deleted) or within CORE, in the early phases of aggregation. The metrics and controls defined by sensor s_{dump} are reported in Table 5.27 and Table 5.28.

The next sensor, $s_{extract}$, is defined over the processing block $extract$ and declares the sub-process metrics $sub\ article\ completeness$, which is evaluated as follows

$$sub\ article\ completeness = \frac{N_{emptyMandatoryFields}}{N_{mandatoryFields}} \tag{5.5}$$

for each XML article processed by the processing component $metadata\ extraction$. It is interesting to monitor whether the process metrics $article\ completeness$, evaluated

Table 5.27: *Metrics defined by sensor s_{dump} over storage block $OAI\ dump$*

Sensor s_{dump}			
Metrics	**Type**	**Description**	**Labels**
total articles	dm	count the number of OAI records contained in the XML dump	repositoryId=*id*

Table 5.28: *Controls defined by sensor s_{dump}*

Sensor s_{dump}			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
total articles	last 3 sessions	repositoryId=*	the number of total articles is monotonic increasing

at the end of each $extraction$ block session by averaging the observations produced by $sub\ article\ completeness$, is above the 80%. The metrics and controls defined by sensor $s_{extract}$ are summarized in Table 5.29 and Table 5.30.

Table 5.29: *Metrics defined by sensor $s_{extract}$ over processing block $extract$*

Sensor $s_{extract}$			
Metrics	**Type**	**Description**	**Labels**
sub article completeness	spm	completeness of each article XML record	repositoryId=*id*
article completeness	pm	$avg(sub\ article\ completeness)$ for the current session	repositoryId=*id*

Table 5.30: *Controls defined by sensor $s_{extract}$*

Sensor $s_{extract}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
article completeness	last session	repositoryId=*	the average article completeness is above a 80% threshold

A third sensor s_{mySql} is defined over the storage block $mySql$ and declares the data metrics $total\ articles$, which counts the number of articles persisted in the database for the repository currently involved in the data flow, as indicated in Table 5.31. As emerged from the monitoring intentions, this metrics is controlled as specified in Table 5.32 ensuring that the number of articles persisted is increasing monotonically over time.

Moving further, we declared a sensor s_{pdfs} over the storage block $PDF\ dump$. The sensor declares the data metrics $total\ PDFs$, which is evaluated by counting the number of PDF files stored in the filesystem for the repository involved in the data flow. It is our interest to verify that the values produced by this metrics are monotonic increasing over time as a negative variation in such a feature indicates that an issue might have occurred either at a repository level (e.g. broken links advertised, missing or corrupted files, etc) or in the CORE's PDF download business logic (e.g. an introduced error in

Table 5.31: *Metrics defined by sensor s_{mySql} over storage block $mySql$*

Sensor s_{mySql}			
Metrics	**Type**	**Description**	**Labels**
total articles	dm	count the number of extracted articles	repositoryId=*id*

Table 5.32: *Controls defined by sensor s_{mySql}*

Sensor s_{mySql}			
	Selector(s)		**Analyzer**
Metrics	p_t	p_l	
total articles	last 3 sessions	repositoryId=*	the number of total articles extracted is monotonic increasing

the algorithm configuration or implementation). The metrics and controls defined by sensor s_{pdfs} are reported in Table 5.33 and Table 5.34.

Table 5.33: *Metrics defined by sensor s_{pdfs} over storage block $PDF\ dump$*

Sensor s_{pdfs}			
Metrics	**Type**	**Description**	**Labels**
total PDFs	dm	count the number of PDF files in a dump folder on the filesystem	repositoryId=*id*

Table 5.34: *Controls defined by sensor s_{pdfs}*

Sensor s_{pdfs}			
	Selector(s)		**Analyzer**
Metrics	p_t	p_l	
total PDFs	last 3 sessions	repositoryId=*	the number of total PDFs downloaded is monotonic increasing

The storage block $fulltext\ dump$ defines a sensor $s_{fulltexts}$, whose declared data metrics $total\ fulltexts$ is reported in Table 5.35. The metrics is evaluated by simply counting the number of full-texts successfully extracted from PDF files and stored in the relevant filesystem location for the repository in exam. The values extracted from the metrics are then controlled in order to ensure that the number of full-texts extracted is monotonic increasing throughout time, as indicated in Table 5.36.

As outlined in Section 2.2, the monitoring intent includes combined checks of metrics sampled in different "locations" within the data flow (i.e. executing at different time); in this case we need to compare observations produced by metrics declared by two different storage blocks. For this reason, we prepared a compound block enclosing the two storage blocks $PDF\ dump$ and $fulltext\ dump$ declaring a sensor $s_{compound}$ enabling the generation of compound metrics and their control. In particular, $s_{compound}$ declares

Table 5.35: *Metrics defined by sensor $s_{fulltexts}$ over storage block fulltext dump*

Sensor $s_{fulltexts}$			
Metrics	**Type**	**Description**	**Labels**
total fulltexts	dm	count the number of full-texts	repositoryId=*id*

Table 5.36: *Controls defined by sensor $s_{fulltexts}$*

Sensor $s_{fulltexts}$			
Selector(s)			**Analyzer**
Metrics	p_t	p_l	
total fulltexts	last 3 sessions	repositoryId=*	the number of total full-texts extracted is monotonic increasing

the compound metrics $fulltexts VSpdfs$ evaluated as the ratio

$$fulltexts VSpdfs = \frac{total\ fulltexts}{total\ PDFs} \tag{5.6}$$

and summarised in Table 5.37. The observations extracted from this metrics are then analyzed in order to check whether the number of full-texts extracted is at least 98% of the number of PDFs successfully downloaded and monotonic increasing over time, as indicated in Table 5.38.

Finally, we want to ensure some controls at data flow session level, combining and comparing observations extracted from different metrics in different parts of the data flow and ensuring that the entire processing has been carried out as expected. First of all, we prepared one last sensor s_{index} over the last storage block $index$. s_{index} declares the metrics reported in Table 5.39 measuring again, by means of queries, the number of articles, articles with PDFs and articles with full-text, this time from the index. As we will see shortly, these metrics serve only for controls at data flow level (e.g. alignment of different backends), hence no control is defined at this moment on this sensor. Then a compound block enclosing all the monitoring flow is defined; over such a block we define a sensor s_{flow}, declaring one compound metrics $pdfs VSarticles$ evaluated as the ratio

$$pdfs VSarticles = \frac{total\ PDFs_{(block=PDF\ \ dump)}}{total\ articles_{(block=index)}} \tag{5.7}$$

as indicated in Table 5.40. Please notice that s_{flow} has visibility on all the metrics declared so far as it inherits the scopes from the blocks enclosed by the compound block declared at data flow level. Over the declared compound metrics and all the metrics inherited by enclosed blocks, we define the controls reported in Table 5.41 mainly checking if observations produced by the compound metrics $pdfs VSarticles$ are monotonic increasing over time and above the 80%, and if the values produced by metrics $total\ articles$, $total\ PDFs$ and $total\ fulltexts$, sampled from different parts of the data flow, are aligned at the end of a data flow execution.

Table 5.37: *Metrics defined by sensor $s_{compound}$*

Sensor $s_{compound}$			
Metrics	**Type**	**Description**	**Labels**
fulltextsVSpdfs	cm	the percentage of PDFs for which a full-text can be extracted	repositoryId=*id*

Table 5.38: *Controls defined by $s_{compound}$*

Sensor $s_{compound}$			
	Selector(s)		**Analyzer**
Metrics	p_t	p_l	
fulltextsVSpdfs	last session	repositoryId=*	the ratio should be at least 98%
fulltextsVSpdfs	last 2 sessions	repositoryId=*	the ratio should monotonic increasing

Table 5.39: *Metrics defined by sensor s_{index} over storage block $index$*

Sensor s_{index}			
Metrics	**Type**	**Description**	**Labels**
total articles	dm	count the number of indexed articles	repositoryId=*id*
total PDFs	dm	count the number of indexed articles with PDF	repositoryId=*id*
total fulltexts	dm	count the number of indexed articles with extracted full-text	repositoryId=*id*

Table 5.40: *Metrics defined by sensor s_{flow}*

Sensor s_{flow}			
Metrics	**Type**	**Description**	**Labels**
pdfsVSarticles	cm	the percentage of articles for which a PDF can be retrieved	repositoryId=*id*

Table 5.41: *Controls defined by s_{flow}*

Sensor s_{flow}			
	Selector(s)		**Analyzer**
Metrics	p_t	p_l	
total articles	last session	repositoryId=*,block=index	check alignment
total articles	last session	repositoryId=*,block=mySQL	
total articles	last session	repositoryId=*,block=OAI dump	
total PDFs	last session	repositoryId=*,block=index	check alignment
total PDFs	last session	repositoryId=*,block=PDF dump	
total fulltexts	last session	repositoryId=*,block=index	check alignment
total fulltexts	last session	repositoryId=*,block=fulltext dump	
pdfsVSarticles	last 2 sessions	repositoryId=*	check whether the ratio is monotonically increasing
pdfsVSarticles	last session	repositoryId=*	check whether the ratio is more than 70%

CHAPTER 6

An Implementation of MoniQ

In this chapter, we discuss the current realization of MoniQ and its soundness and completeness in relation to the reference architecture drawn in Chapter 4. In particular, Section 6.1 describes the current implementation of MoniQ server, its capabilities and integration with the target Data Infrastructure (DI). For this purpose, we introduce MoniQ-java-di, a thin-layer of Java classes developed in order to facilitate the integration of MoniQ into Java-based DIs as the ones already presented in Chapter 2.

In Section 6.2, we finally conclude with a showcase of system's main features via screenshots taken from MoniQ instance monitoring the OpenAIRE production environment. Also, the access to a live demo (with read-only permissions) is provided to the reader.

6.1 Implementation details

In this section, we will focus on the current implementation of MoniQ and put it in relation to the conceptual architecture described in Chapter 4. As already pointed out in Data Flow Quality Monitoring Systems (DFQMSs) requirements (see Section 4.1), MoniQ is an autonomous web application developed in Java, which can be deployed on its own and lives outside the monitored DI. MoniQ server, discussed in Section 6.1.1, offers a deck of Web User Interfaces (WebUIs) for interacting, configuring and consulting the monitoring system. Here, the DI quality manager can define multiple *monitoring scenarios* of interest, i.e. the monitoring flows and the monitoring intents expressed on top of them by means of sensors, metrics and controls.

On the DI side, MoniQ needs to be integrated with the monitored DI in order to receive observations about the defined metrics. As discussed in Section 6.1.2, MoniQ currently provides two ways for being integrated with the DI (see Figure 6.1): (i) a low-level RESTful web API; (ii) an off-the-shelf client-side Java module addressing

Java-developed DIs, called MoniQ-java-di. Once the integration strategy has been chosen, the DI data flow needs to be instrumented in order to accommodate the code taking care of evaluating the metrics of interest and, when needed, handle the signaling coming from MoniQ in order to drive the actuators deployed.

As observations reach MoniQ server, the DI quality manager can inspect their trends over time from the WebUI via interactive time series charts or tabular data, and check in a quick glance the status of the controls defined over them and potential issues present in the infrastructure's data flows. Given a monitoring scenario and the outcome of the defined controls, MoniQ takes care of dispatching *alerts* and *notifications* (e.g. via email) to inform the infrastructure quality manager about the status of the infrastructure and its operation, and driving *actuators* possibly deployed in the monitored DI.

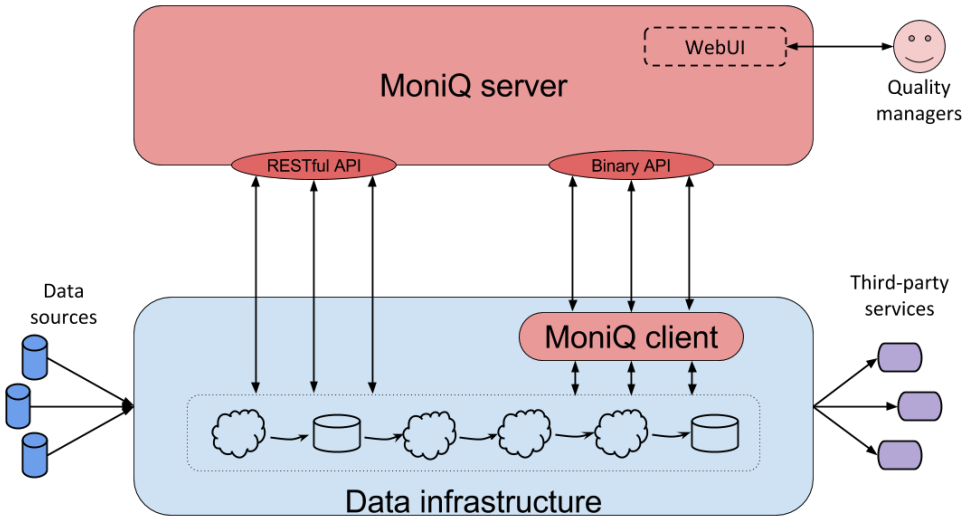


Figure 6.1: *MoniQ implementation*

In the following sections, we describe the current implementation status of MoniQ both on the server side and the DI integration side; the missing features w.r.t. the architecture discussed in Chapter 4 are under development and will be introduced soon.

6.1.1 MoniQ server

The current implementation of MoniQ is not complete with respect to the specification provided in Chapter 4, but includes core elements of the monitoring framework such as m-flows definition, metrics, sensors, and controls, and a WebUI capable of covering a large pool of DI monitoring use-cases. MoniQ's WebUI allows the DI quality manager to design and configure multiple m-flows, which coexist autonomously and in isolation, in terms of storage blocks, processing blocks and monitoring intents defined on top of them. This operation is performed by editing a JSON-formatted configuration based on the specifications of the monitoring flow description language. The DI quality manager configuration preferences are persisted in a relational database (based on PostgreSQL) whose data model is depicted in Figure 6.2. The database holds relations for other entities too such as users, access rights, notification policies and so forth; for the sake of

brevity, these relations are not reported in the figure as non central to the main content of the thesis.

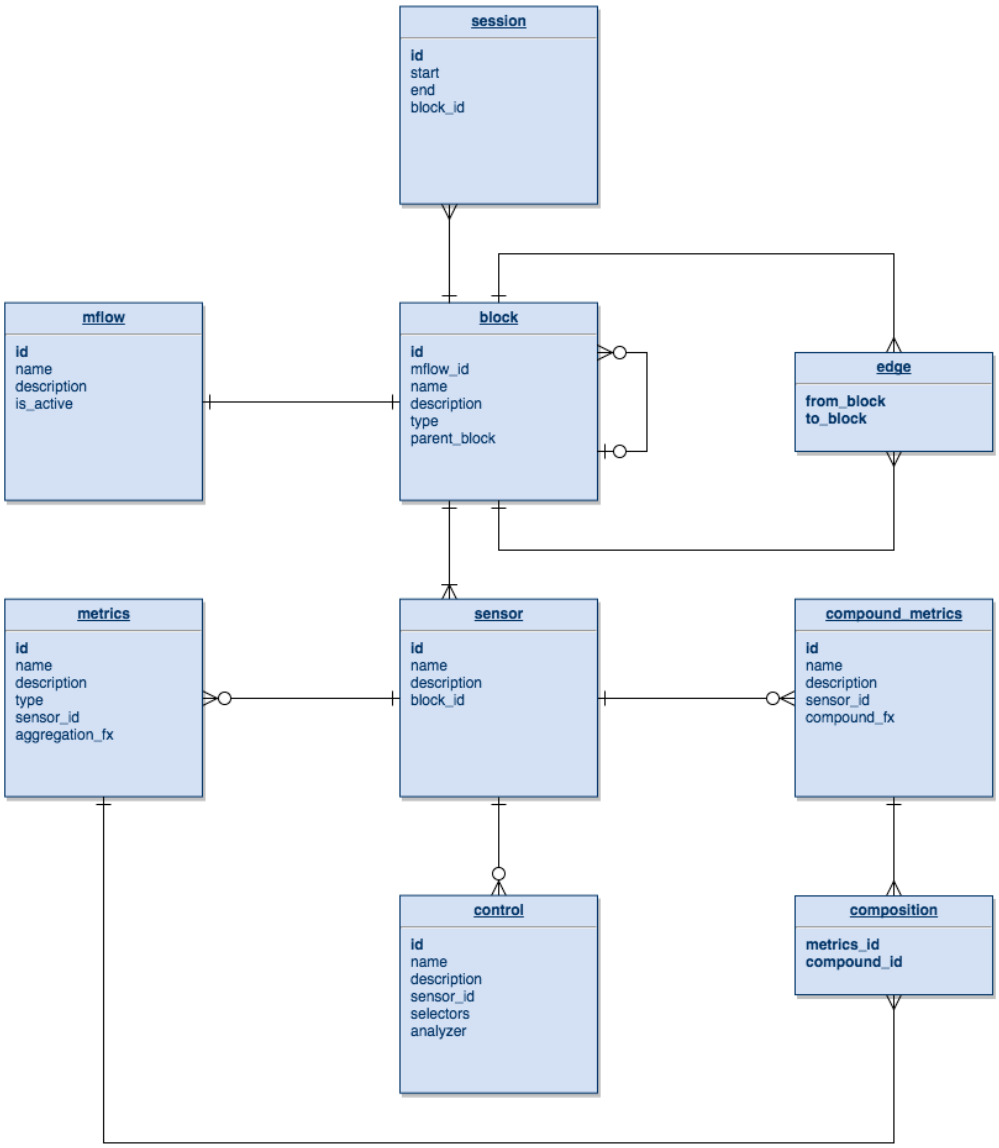


Figure 6.2: *MoniQ data model (relational)*

Although the model already captures the notion of compound blocks (by means of foreign key *parent_block*), the current implementation of MoniQ manages only block-level and m-flow-level sensors. The next software release of the framework, today under development, will also include compound block-level sensors and enhance the WebUI so as to support ability to graphically construct m-flows by composition of processing and storage blocks. The interface will also permit to place sensors over blocks, declare metrics and controls, create compound blocks from the ones defined and compose the metrics within their scopes.

In the following we provide the implementation details of MoniQ server, by delving deeper into the underlying technical choices and rationale.

MoniQ APIs MoniQ server exposes a RESTful API in order to receive observations from the DI; the REST endpoint is mapped onto

`http://moniQ_host:80/<moniQ_context>/rest/observations`

where MoniQ awaits for JSON formatted observations as indicated in Listing 6.1.

Listing 6.1: *Observation format*

```
{
    "time" : <timestamp>,
    "scenario": <scenarioId>,
    "block": <blockId>,
    "sensor": <sensorId>,
    "metrics" : <metricsId>,
    "dfi" : <dataflowId>    // optional
    "labels": {
        <label1_key> : <label1_value>
        //...
    },
    "value": <value>
}
```

The RESTful API exposed by MoniQ also offers an endpoint

`http://moniQ\_host:80/<moniQ\_context>/rest/notifications`

where MoniQ awaits for JSON formatted closing notifications as indicated in Listing 6.2.

Listing 6.2: *Closing notification*

```
{
    "time" : <timestamp>,
    "block": <blockId>,
    "dfi": <dataflowId>     // optional
}
```

Finally, MoniQ server also exposes an API for the communication with MoniQ-java-di module in a reliable and optimized way; such endpoint relies on a binary protocol for web services communication over HTTP realized with Hessian[1].

In relation to the latency overhead consideration discussed when drawing DFQMSs requirements (see Section 4.1), the $t_{com} = t_{send} + t_{receive}$ latency factor is negligible for both the RESTful and the binary API. We run some tests in order to assess the t_{com} factor considering an average sized observation with a handful of labels (more than we ever used so far, for safety) and we ended up in promising results with scarce impact over DI's performances. When the MoniQ-java-di library cannot be used (e.g. when the DI programming language is other than Java) or usage of the RESTful API is preferred, the delay introduced by the communication with MoniQ RESTful API is still contained despite the lack of optimizations provided by Hessian (plain HTTP POST requests).

[1] Hessian binary web service protocol ,`http://hessian.caucho.com`

Observations persistence At the moment, MoniQ takes care of persisting observations either on a dedicated table in the same relational database described above or in a time series database based on InfluxDB[2]; the backend of choice can be specified by the quality manager within MoniQ configurations. For the sake of simplicity and agile prototyping, the first implementation choice for observations persistence fell on the same relational database in use for persisting m-flows as this permitted in our first MoniQ realization to manage just one backend for data flow quality monitoring. The relation storing observations is described as follows:

$$observation(\underline{time}, scenario\_id, block\_id, sensor\_id, dfi, metrics\_id, value, labels)$$

Later, after a re-engineerization and refactoring of MoniQ, the usage of a separated, observations-dedicated and optimized backend has been introduced by adopting InfluxDB. Being a database specially devised to deal with time series, InfluxDB offers out-of-the-box tools for querying, aggregating and managing metrics. Also, it can be easily integrated with off-the-shelf, widely-adopted visualization tools such as Grafana[3] enabling the creation of personalized dashboards in a straightforward way. The data model adopted by InfluxDB for persisting points in time series (i.e. MoniQ observations) is described on his documentation website[4] and follows the model describe below:

$$\begin{aligned}&\langle unix\_timestamp\rangle, \langle metrics\rangle,\\ &\qquad [\langle tag_{k_1}\rangle = \langle tag_{v_1}\rangle, \dots],\\ &\qquad \langle field_{k_1}\rangle = \langle field_{v_1}\rangle [, \langle field_{k_2}\rangle = \langle field_{v_2}\rangle, \dots]\end{aligned}$$

being tags optional and at least one field required. The difference between values and tags regards the fact that tags are indexed by InfluxDB while fields are not; namely, fields are meant to contains data points for the indicated metrics, while tags contain metadata about the points. As an example for an unrelated use case, a suggested usage for the data model adopted by InfluxDB would be:

$$\begin{aligned}&1434067467000000000, temperature,\\ &\qquad room = living\_room,\\ &\qquad season = summer,\\ &\qquad internal = 25, external = 31\end{aligned}$$

In our MoniQ implementation, we mapped observations to the InfluxDB model in the following way:

$$\begin{aligned}&\langle timestamp\rangle, \langle metrics\_id\rangle,\\ &\qquad block = \langle block\_id\rangle,\\ &\qquad sensor = \langle sensor\_id\rangle,\\ &\qquad dfi = \langle dataflow\_id\rangle,\\ &\qquad [\langle labelName_1\rangle = \langle labelValue_1\rangle, \dots],\\ &\qquad value = \langle value\rangle\end{aligned}$$

[2]InfluxDB, `https://www.influxdata.com`
[3]Grafana, `http://grafana.org`
[4]InfluxDB documentation, `https://docs.influxdata.com/influxdb/v1.0/introduction/getting_started`

As InfluxDB is able to manage multiple (time series) databases, one different database is created for each monitoring scenario, and within a scenario each metrics defined in MoniQ maps to a separate time series on InfluxDB with all due tags capturing the context of single observations. The only exception is about processing blocks and sub-process metrics; in this case, whenever a block sessions is spawned, observations generated by the sub-process metrics declared are persisted in temporary time series. Once the block session has terminated (i.e. upon reception of the relative closing notification), MoniQ takes care of aggregating sub-process metrics observations into process metrics observations following the indications specified in the monitoring scenario, i.e. aggregating in accordance to the aggregation function specified in the column $metrics.aggregation\_fx$. Once aggregated, the temporary sub-process metrics are thrashed by MoniQ and the space occupied freed; the same happens if a block session does not close correctly (after a configurable timeout). Similarly, when an entire m-flow session terminates incorrectly (after a timeout) and a new one starts, the collected observations relative to the (entire) m-flow failed session are thrashed.

In relation to the latency overhead consideration discussed when drawing DFQMSs requirements (see Section 4.1), the server-side latency factor t_{handle} generally depends on the backend of choice for persistence of observations. For our implementation we run several tests for both the chosen backends and assessed the suitability of two. InfluxDB claims[5] "support for hundreds of thousands of writes per second" on its documentation web pages; this statement has been verified by our tests confirming t_{handle} as a fraction of a millisecond, a time indeed negligible, at least for our applications. In general, the throughput sustainable by InfluxDB and by PostgreSQL in the current implementation of MoniQ has greatly satisfied our needs during the realization of our use cases.

Controls Controls over observations generated by metrics can be defined in terms of their selectors and analyzer from the WebUI; the current implementation for selectors can narrow down the selection of observations from a metrics according to predicates over labels p_l and a predicate over time p_t. The predicate imposing conditions over observations' values p_v has not been yet introduced. The time predicate p_t has been implemented to enable limits on the number of sessions (back in time from the last available one) to be considered and to be passed to the analyzer. The analyzer of choice can be selected from a drop-down menu; analyzers used by controls come with off-the-shelf implementations for most common comparison functions such as equality/alignment, upper and lower bound, peak/valley detection, threshold, percent variation, and so forth. The semantic of an analyzer drives the DI quality manager into defining compatible selectors appropriately via the WebUI: some analyzers require only one selector, while others require two or more selectors, whose order may or may not be relevant for the analysis. In any case, the collection of analyzers can be extended at any time by the DI quality manager requiring more specific behaviours.

As an example, Figure 6.3 describes a control checking whether two metrics generated observations with the same trend. The control setup selects observations generated by metrics $pdfs\_count$ during last three sessions and having the label *block* equal to *PDF dump* and *repositoryId* equal to a wildcard expression *?id*, meaning that the control has to be verified for every value of the label *repositoryId*. The other selector does the same

[5] InfluxDB, `http://influxdata.com/time-series-platform/influxdb`

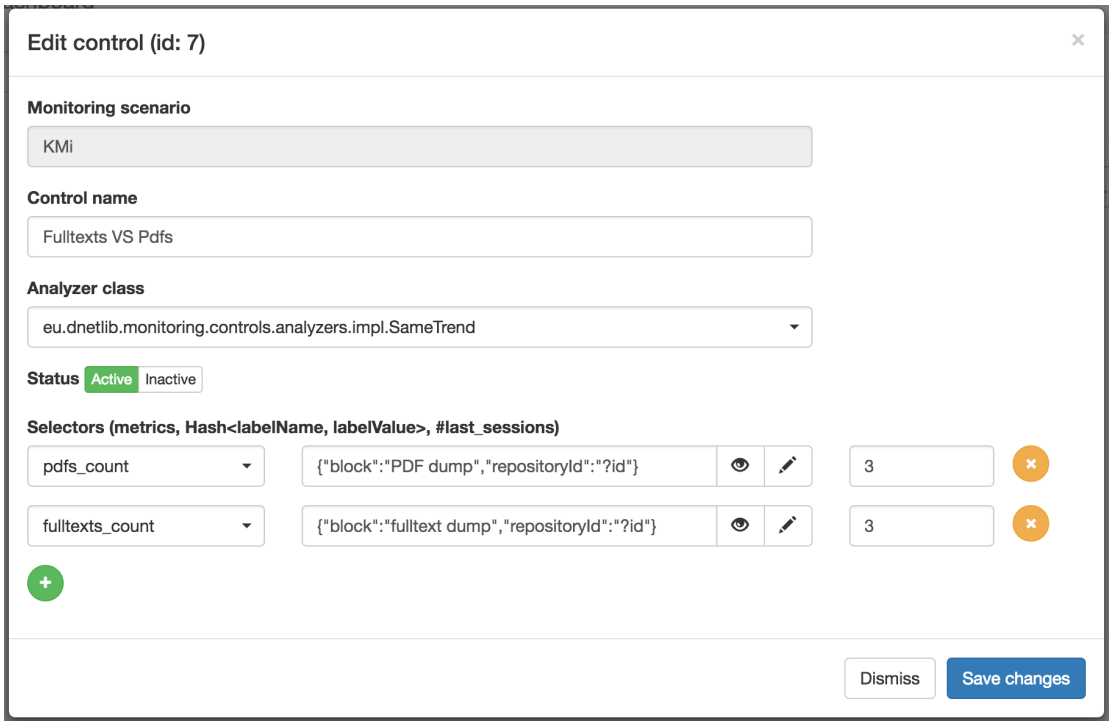


Figure 6.3: *Screenshot of a control configuration in MoniQ*

for metrics *fulltext_count*, but selects observations with *block* set to *fulltext_dump* and *repositoryId* equal to the same wildcard expression *?id*. Finally, the control specifies that the selected observations via the two selectors must be passed to an analyzer that checks if the two time series have the same trend throughout time, i.e. last 3 sessions, as specified by the time predicate. Since this analyzer can work with two or more selectors the interface display a widget that enables the quality manager to specify as many selectors as he needs. The proposed example also shows the usage of the so called *template control engine* offered by MoniQ, which simplifies and reduces the time effort required during the definition phase of monitoring intents, discussed in Section 4.5.

It is worth mentioning that the implementation of the time predicate has to be extended in order to select observations from a give date to present or within a provided date range, as specified by the drawn reference architecture. A possible selection strategy is about taking into account only the observations belonging to m-flow sessions that started and completed entirely *within* the provided time range, i.e. if one of the specified time boundaries "crosses" an m-flow session not yet terminated, this session is not taken into consideration.

Reporting MoniQ enables the generation of an exhaustive *report* about the defined metrics and controls providing, via the WebUI, insight in a quick glance about key features and potential issues present in the DI data flows. In particular, at the time of writing, MoniQ WebUI enables a *by-metrics inspection*, i.e. selecting a metrics of interest first and then visualize its trend and the status of controls in which the metrics is involved, and a *by-control inspection*, i.e. selecting a control on interest first and then visualize the trends of all the trends of the metrics involved in the check. From the

WebUI, the quality manager can plot the trends for the metrics of interest both as time series charts and as tables, explore and zoom in and out the graph, and visualize punctual values and metadata of each observation.

Alerts & notifications Given a set of controls, MoniQ also takes care of raising *alerts* and *notifications* informing the DI quality manager about the status of the infrastructure and its operation. At the time of writing, MoniQ evaluates controls as soon as possible, i.e. whenever a block session or an m-flow session is closed, and dispatch an alert whenever the check does not pass.

6.1.2 Data infrastructure integration

In order to ensure the declared monitoring intents, MoniQ requires DI quality managers to instrument the source code of DI data flows in order to deploy *sensor hooks* and start evaluating the selected metrics and export observations to MoniQ server. The source code instrumentation strategy is a common technique used also by other monitoring approaches such as Prometheus, which offers client libraries for various languages[6], or any other log-based application monitoring, e.g. applications monitored with ELK or TICK stacks, where log traces can be exported and parsed in order to extract metrics useful during troubleshooting.

When the DI quality manager decides to integrate MoniQ using the RESTful web API provided by MoniQ server, he has to take care of invoking it opportunely in order to send observations and closing notifications to the right sensor and block declared in the m-flow. The RESTful endpoint receiving the observations is described in Section 6.1.1 and can be contacted with plain HTTP requests. As already remarked in Section 4.1, the DI quality manager is in charge of providing the implementation of metrics.

The current implementation of MoniQ also includes an off-the-shelf client-side Java module addressing Java-based DIs and Java-based sub-parts, services or components of mixed-technology DIs. Such a module, MoniQ-java-di for short, provides (i) a pre-canned thin layer of Java classes hiding the complexity due to the communication towards MoniQ server for sending generated observations and closing notifications, and (ii) a small catalogue of ready-to-use, prior configuration, basic metrics implementations capable of working with common situations encountered during the application of MoniQ in our use cases, which are however general and applicable broad-spectrum in other contexts too. MoniQ-java-di relies on the binary API based on Hessian exposed by MoniQ server for its communication.

In the following, we will provide a level of details about MoniQ-java-di sufficient to grasp the rationale behind it without delving into fully-fledged implementation details[7]. As a first implementation, MoniQ-java-di would provide two base abstract classes, `DataMetrics` and `ProcessMetrics`, providing an abstract method `produceValue()` each and embedding the logic and boilerplate code for the communication with MoniQ server (i.e. Hessian protocol client) and creation of observations from single values obtained by metrics application. In this case, custom metrics can be realized by deriving in hierarchy such classes and implementing the method

[6] Prometheus source code instrumentation, `http://prometheus.io/docs/instrumenting/clientlibs`

[7] The source code, however, is available at `https://svn.driver.research-infrastructures.eu/driver/data-flow-monitoring`

Listing 6.3: *DataMetrics and ProcessMetrics*

```
abstract class DataMetrics extends AbstractMetrics {
    //...
    protected abstract Double produceValue();
}

abstract class ProcessMetrics extends AbstractMetrics {
    //...
    protected abstract Double produceValue(Class<? extends
        Object>... data);
}
```

`produceValue()` containing the business logic for metrics evaluation. In the case of data metrics, the method computes the evaluation over a collection persisted on a backend of reference, e.g. by lunching a count query and returning the numeric result; in the case of a process metrics, the metrics is instead evaluated on `data` objects passed to the method. The developer completes the customization of the metrics by referencing to which block/sensor in the m-flow the produced value is relevant.

The implementation just described perfectly fits the requirements drawn by the MoniQ reference architecture. However, despite the soundness, it carries some technical suboptimality that can be improved. In the following, we discuss three further refinements we introduced within MoniQ-client-di.

Metrics behaviour factorization This basic implementation, whenever multiple metrics must be evaluated against the same collection (in a storage component) or the same data (flowing in a processing component), implies that every single metrics implementation has to repeat portions of code either to access the collection or the data to analyze. For example, if multiple metrics must be evaluated against a collection stored in Solr, each one of the metrics would have to instantiate a client for Solr, establish a connection and run a query; similarly, each metrics to be evaluated over a relational database would have to embed the connection layer (e.g. JDBC). As another example, if multiple metrics must be evaluated against an XML file, each one of the metrics would have to include an XML parser, parse the file and run his xQueries.

It is clear how this implementation suffers from common technical drawbacks (e.g. boilerplate code, suboptimal resource allocation, etc.) and could benefit from some factorization. For this reason, MoniQ-java-di lets data metrics executing queries with the same semantic over the same collection to share the same core business logic by offering the class `DataMetricsExecutor`. The DI quality manager can extend it in hierarchy and implement the required methods to code the business logic of the executor: a method `initialize()` for setting up the executor, a method for closing down and cleaning resources `shutdown()` and a generic method `produceValue(Map<String, String> metricsParams)` in charge of performing the measure. The variable `metricsParams` contains all the necessary for the metrics to execute its business logic (e.g. query string, target collection, etc.). Similarly, MoniQ-java-di provides a class `ProcessMetricsExecutor` covering the dual use case involving process metrics.

Listing 6.4: *DataMetricsExecutor and ProcessMetricsExecutor*

```
class DataMetricsExecutor extends AbstractMetrics {
    //...
    protected abstract void initialize();
    protected abstract void shutdown();
    protected abstract Double produceValue(Map<String,String>
        metricsParams);
}

class ProcessMetricsExecutor extends AbstractMetrics{
    //...
    protected abstract void initialize();
    protected abstract void shutdown();
    protected abstract List<Double> produceValues(Class<?
        extends Object>... data);
}
```

Metrics behaviour/configuration decoupling Despite the enhancement previously discussed, with the current solution the behavior and configuration of the metrics are *hard-coded* into its implementation. This has however a drawback: even the slightest modification of its configuration would imply a modification in the metrics implementation and, subsequently, a stop and redeploy of the component housing the sensor hook. As an example, take into consideration the already introduced $compliance$ metrics checking to which extent a set of XML fields complies with defined vocabularies. The behavior of the metrics is outlined by the operation $\frac{N_{compliantFields}}{N_{fields}}$. Imagine now that the DI quality manager wants to add to the set a new field to be checked; despite the metrics implementation is ready to cope with the new requirement, the quality manager would have to change it in order to accommodate the new field, i.e. a new xPath to check.

In order to avoid such an inconvenient, MoniQ offers a centralized storage service for metrics configurations and enables the quality manager to store and modify metrics parameters via WebUI so that the metrics implementation can retrieve the most updated parametrization at runtime from MoniQ server. Of course, even this mechanism cannot avoid a modification of the metrics implementation whenever a change in the behaviour is needed. To introduce such a modification, the schema of the database has been modified in order to persist metrics configuration parameters expressed as a JSON object.

$$sensor(\underline{id}, name, description, block\_id, conf)$$
$$metrics(\underline{id}, name, description, type, sensor\_id, aggregation\_fx, labels, conf)$$

Template metrics configuration Even the last proposed enhancement can be further improved. Consider the CORE use case; the quality manager has to evaluate, for every repository registered to the DI, the number of $total\ articles$ on a backend (i.e. Elasticsearch in this case). With the implementation explored so far, the quality manager would prepare a `DataMetricsExecutor` capable of running a batch of generic queries against a Elasticsearch endpoint and initialize it at runtime with an appropriate configuration containing the queries. However, a query contained in the configuration would be

dedicated to the evaluation of the metrics just for one of the thousands repositories in CORE and the DI quality manager would have to replicate the configuration for each one of them and alter the query clause selecting over repositories.

As this practice is scarcely maintainable, we enabled the creation of *template metrics configurations* in which variables can be declared as placeholders in the configuration – e.g. $?variable$ – and have to be filled out later on, at runtime, by the code invoking the metrics. In our example, the quality manager has to edit a configuration, write the query clause as $repositoryId = ?id$ and, at runtime, provide to the template engine a map resolving all the declared placeholders.

MoniQ-java-di also offers a small catalogue of ready-to-use, prior configuration, metrics implementations capable to work with the common cases encountered during the implementation of our use cases, i.e. metrics working with XML records, PDF and CSV files and metrics executors working over collections persisted in Solr, elasticsearch, redis and postgres.

Finally, MoniQ-java-di provides an utility class for sending closing notifications when appropriate.

Listing 6.5: *Closing notification*

```
class ClosingNotification {
    //...
    public static void notify(String blockId, String dataflowId){
        //...
    }
}
```

6.2 Showcase

The images proposed in this section are taken from the MoniQ instance monitoring the OpenAIRE production environment. A subset of functionalities (mainly read-only) offered by MoniQ monitoring system and metrics exported from the DI can be accessed with the credentials reported in Table 6.1. Here, the guest user can explore the stored metrics (visible for guest inspection) and visualize the controls defined over such metrics and their outcomes.

Table 6.1: *MoniQ showcase instance*

URL	`https://services.openaire.eu/dfm/dashboard.html`
Username	guest
Password	guest

For the sake of space, we selected for demonstration a representative set of showcases taken from the publishing data flow monitoring scenario, described in Section 5.1.4 and reported for completeness in Figure 6.4. The integration of s_{toOai}, s_{oai} and s_{lod} sensors has been introduced recently into the OpenAIRE development environment

(nightly build, developers access only) for testing and fine tuning before being ported to OpenAIRE production environment (accessible to general public). Thus in this showcase, the data displayed is limited to s_{index} and s_{stats} only.

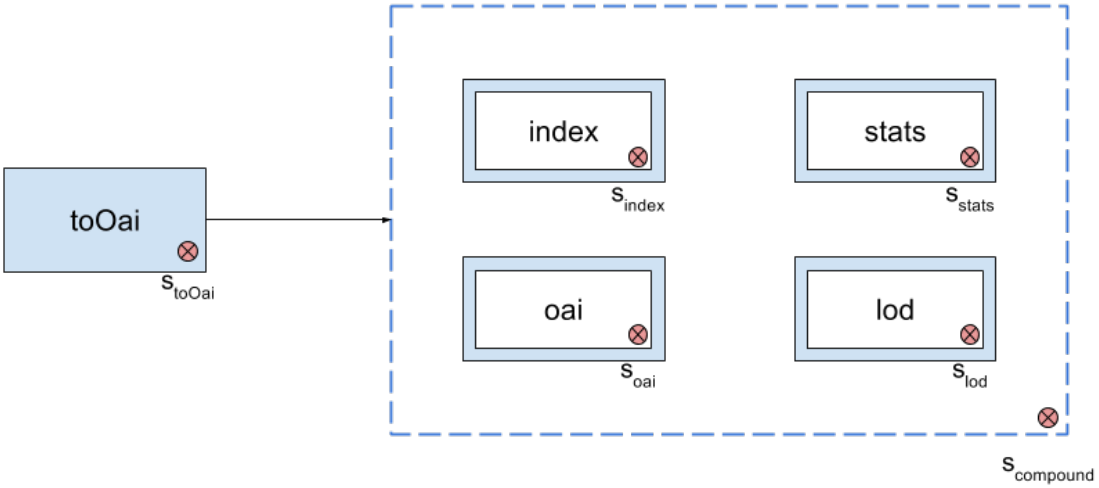


Figure 6.4: *OpenAIRE publishing monitoring flow and monitoring intents*

Figure 6.5 shows metrics *publications* declared by sensors s_{index} and s_{stats} from *index* and *stats* blocks respectively and controlled with a control defined on $s_{compound}$. As we can see the values of the two time series are perfectly matching in the last session, hence the control checking for alignment of the two metrics passes, as represented in the green box on the right. However, the past history of the two metrics shows how these two trends diverged in a couple of occasions; back then MoniQ helped our team in detecting the issue and promptly recover from the transient issue, before the data could be switched to public access. Before the adoption of MoniQ monitoring system, such a control was performed manually (for all the metrics of interest) with consequent error-prone tendency. The second control checks whether the last three observations taken on Redis are monotonic increasing with no "bumps" (meaning the percent variation between two subsequent observations should not exceed 15%); as this condition is verified, the other control passes too (the green box on the left).

Figure 6.6 shows the trend throughout time of the metrics *FP7 publications* declared by sensors s_{index} and s_{stats} from *index* and *stats* blocks respectively, and controlled with a control defined on $s_{compound}$. This time two controls are show in red, meaning that they have failed to pass. In particular, the first one fails to ensure that the two metrics are aligned as can be seen from the tooltip in the figure. The second control checks whether the total number of *FP7 publications* matches the number of *FP7 pubs in 2007 − 2015*, another metrics counting the number of FP7-funded publication correctly dated within the date range of pertinence for FP7 funding stream; as the numbers are not equal, the control is marked in red. This can be noticed as well when inspecting the metrics *FP7 pubs in 2007 − 2015*, as shown in Figure 6.7. A by-control inspection over the failing control, selected for example from the global report view shown in Figure 6.8, would show the comparative chart pictured in Figure 6.9.

Figure 6.10, shows the trend of the metrics *missing titles*, accounting the number of publication records having no titles. The metrics is expected to be monotonic decreasing

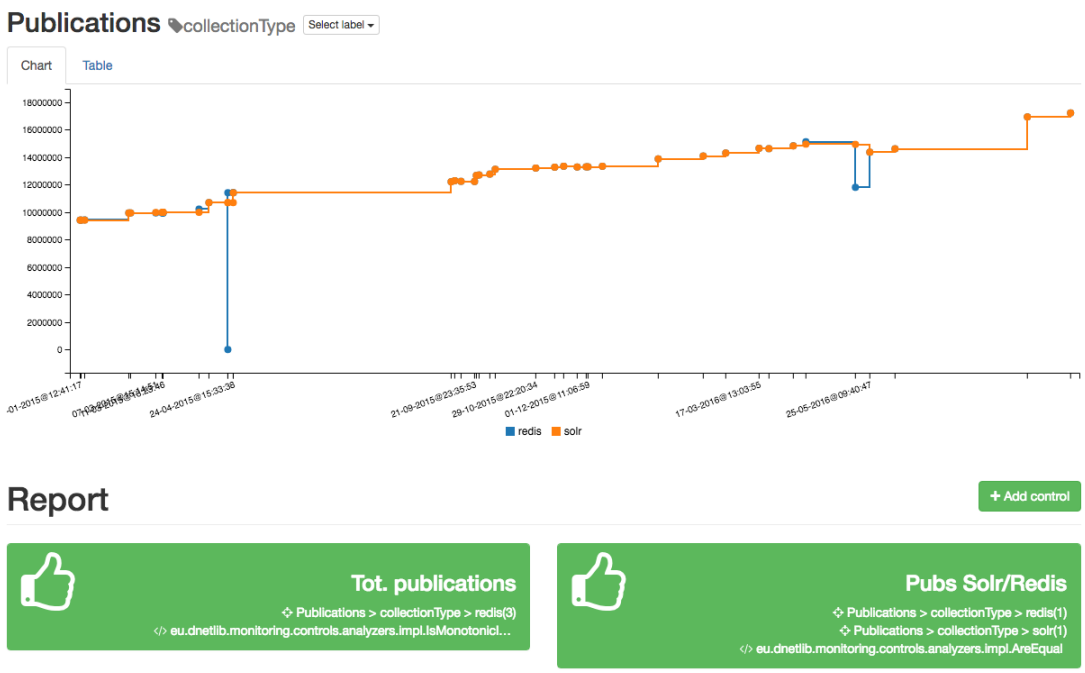


Figure 6.5: *Screenshot of the MoniQ dashboard for the metrics "publications"*

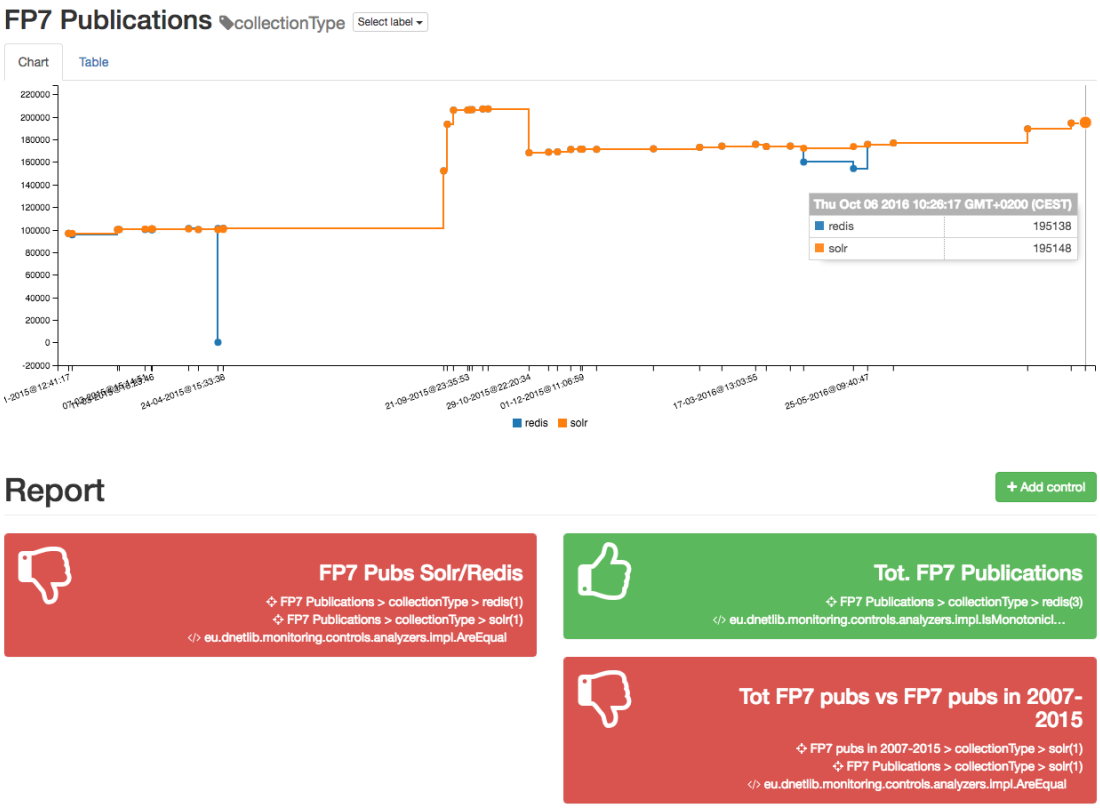


Figure 6.6: *Screenshot of the MoniQ dashboard for the metrics FP7 publications*

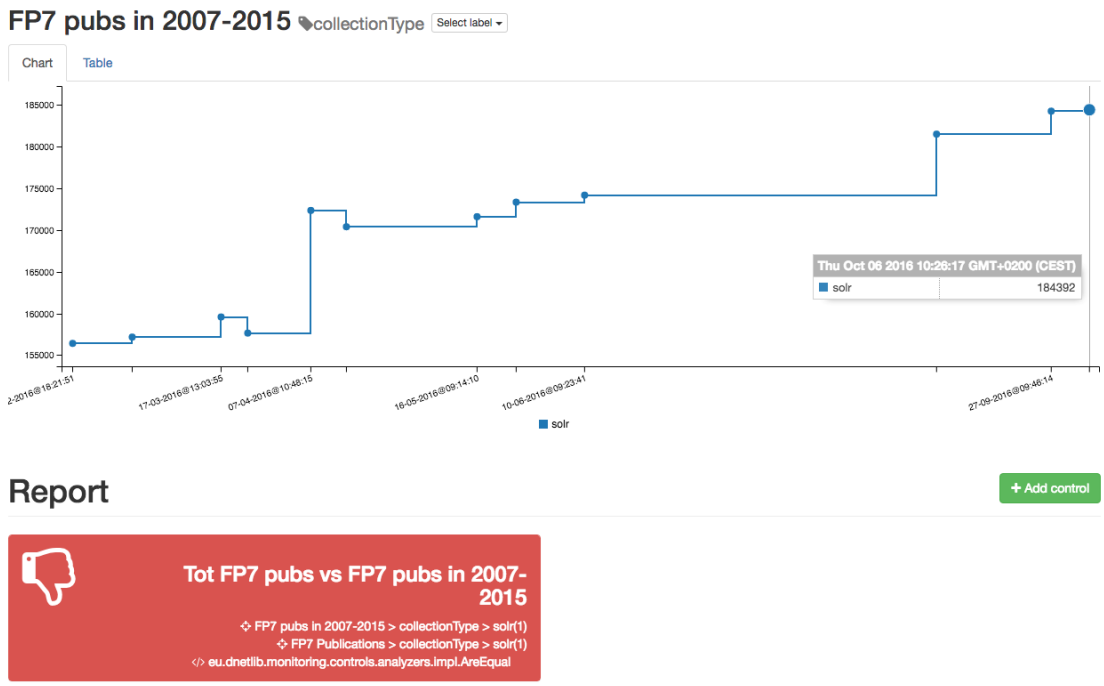


Figure 6.7: *Screenshot of the MoniQ dashboard for the metrics FP7 pubs in 2007 – 2015*

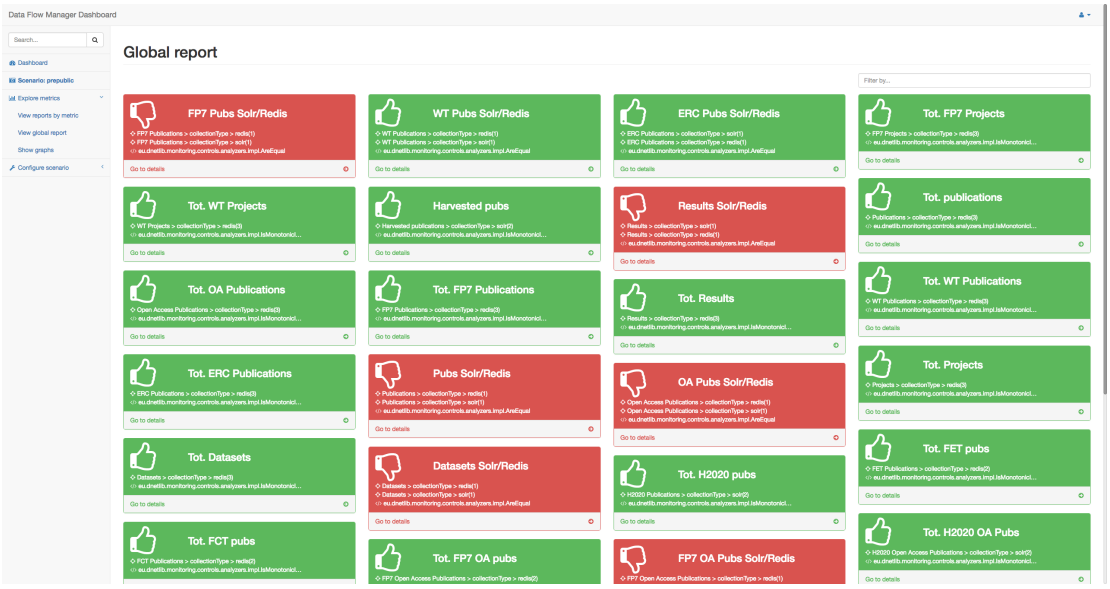


Figure 6.8: *Screenshot of the MoniQ dashboard visualizing the global report about the status of the infrastructure w.r.t. the defined controls*

and hopefully zero, however in last two sessions the value jumped from 5879 yo 5890 and therefore the controls fails.

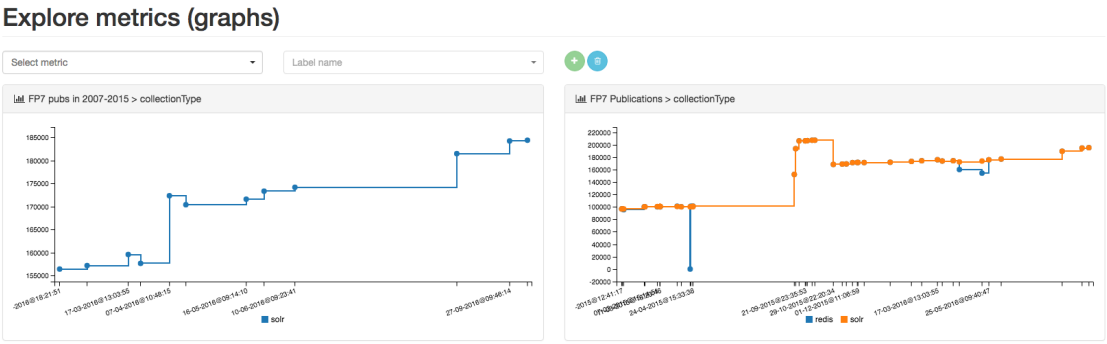


Figure 6.9: *Screenshot of the MoniQ dashboard visualizing the details of a control*

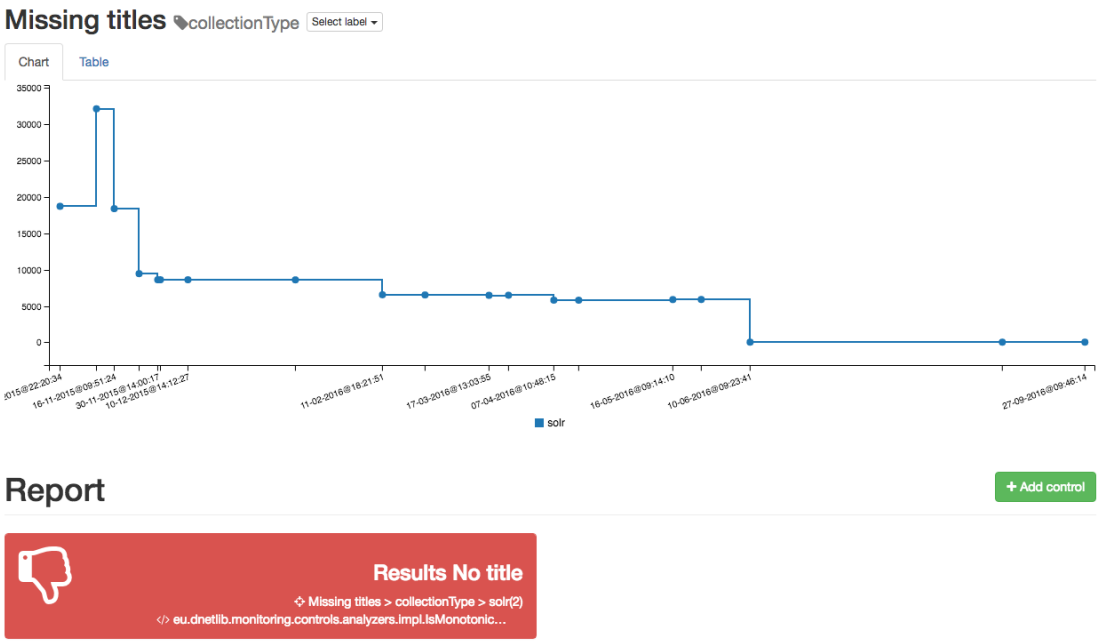


Figure 6.10: *Screenshot of the MoniQ dashboard for the metrics missing titles*

CHAPTER 7

Conclusions

The motivation driving the research presented in this thesis grounds in the lack of support by current tools available on the market and in the literature to thoroughly target Data Flow Quality Monitoring (DFQM) in Data Infrastructures (DIs), i.e. the assessment over time of the "expected behaviour" of running workflows in the DI in terms of "data quality" and "processing quality" against previously established criteria. As we pointed out in the introduction, such activity is crucial both for the customers served by the DI, in terms of satisfaction and accountability, and for the DI itself, in terms of better resource allocation, insight of the internals and sustainability. Yet, it is a challenge largely unsolved and exacerbated by the intrinsically multi-faceted and subjective nature of data quality and quality in general [15, 87, 88, 120] and by the generally complex interactions among processing components and data component within data flows in DIs.

Driven by an extensive literature review in the field of data quality and workflow monitoring, and by the study of two real-world use cases (i.e. OpenAIRE and CORE Data Infrastructures) and their related monitoring challenges, we introduced MoniQ, a system enabling Data Flow Quality Monitoring in DIs. We outlined its desired requirements, providing a side-by-side comparison with the approaches found in the literature using the taxonomy introduced, and drawn its architecture in Chapter 4. The specifications of MoniQ architecture provide a *monitoring description language* capable of:

- describing monitoring flows: DI quality managers can express the semantic and the time ordering of their observational intents and capture the essence of the DI data flows to be monitored, by means of *monitoring flow description* primitives;
- describing monitoring intents over the monitoring flows: DI quality managers can specify the metrics and controls required to perform the monitoring using *monitoring intent description* primitives.

MoniQ enables the collection and organization of observations extracted by metrics from DI data flows. Such observations, persisted as time series, can be inspected and automatically queried in order to ensure used-defined constraints and controls, and provide insight about the current and past behavior of the infrastructure. The quality manager can also be provided with alerts and notifications as soon as the monitored DI data flows encounter anomalies. MoniQ is also equipped with a mechanism for providing feedback to the monitored infrastructure in order to trigger a prompt reaction to anomalies, steer DI data flows and attempt, to some extent, an automatic correction of the misbehaviour.

The monitoring system resulting from the implementation of MoniQ specifications is currently employed to monitor the production environment of the OpenAIRE infrastructure and helps its quality managers to promptly spot anomalies and prevent the dissemination of erroneous data to the general public as well as imprecise statistics to EU commission. Prior to the adoption of MoniQ, OpenAIRE production environment took advantage of a more limited kind of monitoring mainly focused on the aggregation and publishing workflows only, and on the comparison of counts among different backends, on which metrics were evaluated manually and thus in a non-systematic, automatic and integrated way. Despite this, the number of metrics to verify was already substantial – about twenty over the four distinct backends and a couple of composite ones – and keeping historical data of the monitored scenarios availing of spreadsheets turned out to be unfeasible in the long run. First of all such a manual practice was indeed prone to human errors and hardly maintainable throughout time, and secondly it was limited just to the exploration of data components participating to the data flow. Our experimental study carried out for the CORE infrastructure revealed a similar *modus operandi* affected by the same flaws and limitations.

With MoniQ instead, given a monitoring flow and its monitoring intent specifications, the monitoring system takes care of persisting observations extracted from both data and processing components of a data flow, aggregating them and dealing with their compositions, and dynamically controlling the resulting time series over time against a set of constraints defined by the quality manager. The adoption of MoniQ made possible to extend the monitoring of data flows also to the other scenarios described in this work and increase the number of metrics being monitored without significantly undermining the management costs for data flow quality monitoring and without showing scalability issues in our real case applications.

7.1 Future work

Our approach can be extended and improved in several directions; in this section we discuss current areas of improvement for the current approach and propose further research directions.

7.1.1 Scalability issues: when “monitoring data” grow big

In general, DIs route and process high loads of data and performing Data Flow Quality Monitoring over such systems eventually ends up in amassing observations until a “big monitoring data” condition is reached. Despite the space occupied by single observations is negligible as the storage engine is optimized and compressed, high-throughput DI

data flows can easily saturate the disk space dedicated to observation persistence if no countermeasure is taken.

The possible solutions to this scalability issue are about (i) *downsampling* the data, i.e. aggregating/summarizing old observations as time goes by; (ii) operating *retention policies*, i.e. keeping observations for a certain amount of time and then freeing space; (iii) a combination of the two strategies, e.g. keeping the high precision raw data only for a limited time, and storing the lower precision, summarized data for much longer or indefinitely. In particular, we identified two viable directions for MoniQ in order to integrate downsampling and retention policies; the two approaches are discussed below.

Quality-manager-driven fine tuning A possible strategy is about exposing mechanisms for the fine tuning of downsampling and retention policies directly to quality managers from MoniQ Web User Interface (WebUI). Retention policies and downsampling of course are tightly bound to the controls that are intended to be ensured over observations and should be configured accordingly. If, for example, observations older than two weeks up to one month get downsampled, then the controls, if any, must be declared having in mind the retention policy in act within the time frame relevant for them.

MoniQ-driven optimization Another possible strategy is to let MoniQ drive the configuration of downsampling and retention policies by taking as input the current controls setup and inferring the optimal configuration in order to guarantee the best storage occupation given the actual monitoring intents. For example, if no control is set to work with observation older than n weeks (or older than m data flow sessions), than the retention policy can be configured automatically in accordance.

7.1.2 Dynamically reconfigured controls

In MoniQ, the configuration of controls is dynamic, i.e. it can be changed from the WebUI at any time, but once configured the configuration is applied until next reconfiguration occurs, which has to happen manually from the quality manager. For example, if a threshold in a controls is set to a value, there is no way to change the current value other than manually via the WebUI. However, in some situations, it could be useful to dynamically adapt the threshold in a programmatic way. As an example, let us assume that, in a similar situation to the one described in Section 5.1.2, a control checks whether the value v of the last observation of the metrics m is within the $\pm 5\%$ of a threshold th. Let us suppose that the last control evaluation resulted in $v = th + 1\%$, thus satisfying the control. At this point, the control could be reconfigured automatically by MoniQ in order to adopt $v = th' \pm 5\% = (th + 1\%) \pm 5\%$ as the logic to use next.

This is just one of the possibilities for dynamic adaptation of MoniQ controls, but of course other use cases could be found, e.g. adapt to a new threshold in function of the values of last n observations. This mechanism however could expose the control framework to unwanted drifts and therefore should be designed and used carefully.

7.1.3 Multi-state controls outcome

In our approach and implementation with opted for controls resulting in a dichotomous outcome (i.e. either passing or failing). However, in some situations, *multi-state* out-

comes, i.e. with different "degrees" of success and failure, could be more useful and provide a better knowledge about the specific control and the situation of DI's internals.

In the literature review reported in Section 3.3, we have already analyzed some approaches that take advantage of this aspect. In [74], the Kepler Scientific Workflow Management System (SWfMS) is instrumented in order to display a three-state visual clue about the data flowing during a workflow execution. Such a clue is provided via a visual decoration (i.e. a blinking, coloured indicator) introduced on top of workflow activities within the workflow execution dashboard. Depending on the color displayed by the indicators (i.e. green/yellow/red) the related data quality feature assumes an acceptable/suspicious/unacceptable value for the data ad hand, and the scientist can judge how the scientific workflow is performing and react consequently. In the approach proposed in [69–71], quality-aware workflows executed in Taverna are capable of classifying data into different quality classes (two or more) and executes quality actions accordingly. This can be used for example for filtering "good data" from "bad ones" (i.e. good or bad for the intended purpose) or associate to quality classes different corrective actions or sub-workflows.

In our framework, a control evaluation resulting in a n-state outcome could be leveraged in order to differentiate the alerting level and notification strategy associated to an anomaly. A "severe alert", for example, could be used to send promptly an email to DI quality managers or alert them on the pager, while a "mild alert" could be stored in a queue and sent as a aggregated notification once per day. Similarly, an issue classified as a "warning" could be notified only when observations produced by a metrics persist out of the admitted range for a configurable time T. Finally, the modification could affect positively also the possible actuators deployed in the infrastructure, which could be triggered with different policies according to the severity of the anomaly.

7.1.4 Real-time control of sub-process metrics

So far, as described in Chapter 4, observations generated by sub-process metrics flow to MoniQ as the processing component executes and, when the session is closed upon reception of a closing notification, they are aggregated into a process metrics according an aggregation function. In the current specification of the framework, controls can be specified only against observations produced by process metrics, while observations generated by sub-process metrics do not surface to the control engine offered by MoniQ.

However, in some cases it could be useful to monitor and control observations produced by a sub-process metrics as the session is still open, i.e. as the observations are being produced and prior their aggregation. For example, consider a simple data flow working in the field of image recognition, where images from a dataset are indexed using the BoF (Bag of Features) approach [100] against a previously computed "dictionary" of visual local features [16, 54] evaluated from a subset of images of the original dataset. As indexing millions of images is a time-consuming operation, suppose that, in order to avoid poor resource allocation, the quality manager wants to ensure that the execution is aborted (e.g. by triggering an actuator) if the average "distance" between the visual local features represented in the dictionary and visual local features extracted from images indexed so far is below a user-defined threshold, meaning that the feature dictionary created is representative for the dataset at hand. Presently, the control would take place right after the session is closed and observations produced by the sub-process metrics

aggregated, hence after the processing component has consumed the entire input dataset. In this situation it would be impossible to trigger the relevant actuator before the block session ends and thus save useless computation cycles in a condition being already proven unsatisfactory.

Hence, extending MoniQ to enable such a real-time control of sub-process metrics is definitely desirable; the control of partial observations produced by sub-process metrics could be triggered by MoniQ either every N observations or every period T and perform a partial aggregation as indicated in the monitoring intent. Despite it is possible to perform such aggregation and control every new observation produced, this could indeed result in an overkill in some cases.

7.1.5 Data analytics of "monitoring data"

The corpus of "data flow quality monitoring data" (in the following, *monitoring data* for short) generated and stored by DFQMSs opens up to new possibilities for data analytics. It is theoretically possible to analyze and mine such data corpus with different intents. A possible research direction is about understanding whether the stored monitoring data can enable *proactive monitoring*. A number of questions rises. Are there application contexts in which monitoring data show some *seasonality*? Are there application contexts showing some kind of *pattern* in generated monitoring data? Is it possible to capture such features in monitoring data? If so, is it possible to take advantage of such information in order to predict or anticipate future possible anomalies and issues in the DI data flows? To which extent and under which conditions? All these questions stimulate our curiosity but, for the time being, remain largely unanswered.

To our knowledge, the only approach that copes with anomaly forecast based on collected monitoring data is Stampede [47, 112], although in the context of workflow monitoring. Stampede leverages machine learning algorithms to analyze workflows' behaviour patterns in order to predict failures[1]; we envisage that some of these techniques can be exploited in our application field too.

7.1.6 Off-the-shelf instantiation and customization of MoniQ

The approach discussed in this thesis is generic and adoptable broad-spectrum with no constraint about underlying technology or type of data/workflows treated by the monitored DI. Similarly, metrics and controls are on purpose conceived for being generic and extensible to radically different needs dictated by the application context.

Nonetheless, it would be possible to investigate further the integrability of MoniQ approach within the Workflow Execution Plan (WEP) [53] – the term "workflow enactment" can also be found in the literature – of a Workflow Management System (WfMS) (or SWfMS) of choice such as Taverna, Kepler or Galaxy. Such an integration would make possible to deliver a ready-to-use instance of MoniQ along with the WfMS as it happens, for example, to Taverna's Provenance plugin[2].

Similarly, it would be possible to study the target community that a MoniQ instance shall serve and the type of workflows and data types it shall deal with in order to finally provide a customized *ad-hoc* instantiation of MoniQ for a specific context of operation.

[1] Poster. STAMPEDE: A Framework for Monitoring and Troubleshooting of Large-Scale Applications on National Cyberinfrastructure. `https://scitech.isi.edu/presentations/2011/stampede-tg11-poster-final.pdf`

[2] Taverna provenance plugin, `http://www.taverna.org.uk/documentation/taverna-2-x/provenance`

The MoniQ customization would have a pre-instantiate metrics catalogue in order to comply with the most common use cases for the research community and use cases being served. Typical metrics could be implemented and provided out-of-the-box as well as most common controls and aggregation operators, resulting in an easily adoptable and configurable data flow monitoring tool specially devised for one particular context.

Bibliography

[1] J. Adelman, M. Baak, N. Boelaert, M. D'Onofrio, J. A. Frost, C. Guyot, M. Hauschild, A. Hoecker, K. J. C. Leney, E. Lytken, M. Martinez-Perez, J. Masik, A. M. Nairz, P. U. E. Onyisi, S. Roe, S. Schaetzel, and M. G. Wilson. ATLAS offline data quality monitoring. *Journal of Physics: Conference Series*, 219(4):042018, 2010.

[2] J. Akoka, L. Berti-Équille, O. Boucelma, M. Bouzeghoub, I. Comyn-Wattiau, M. Cosquer, V. Goasdoué-Thion, Z. Kedad, S. Nugier, V. Peralta, and S. Sisaid-Cherfi. A framework for quality evaluation in data integration systems. *Proceedings of the 9th International Conference on Enterprise Information Systems*, pages 170–175, 2007.

[3] 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 on the 16th International Conference on Scientific and Statistical Database Management*, volume I, pages 423–424, 2004.

[4] Michele Artini, Alessia Bardi, Federico Biagini, Franca Debole, Sandro La Bruzzo, Paolo Manghi, Marko Mikulicic, Pasquale Savino, and Franco Zoppi. The creation of the European Film Archive : achieving interoperability and data quality. In *8th Italian Research Conference on digital Libraries*, pages 1–12, 2012.

[5] Claudio Atzori. *gDup: an integrated and scalable graph deduplication system.* PhD thesis, University of Pisa, 2016.

[6] Donald Ballou, Richard Wang, Harold Pazer, and Giri Kumar.Tayi. Modeling information manufacturing systems to determine information product quality. *Management Science*, 44(4):462–484, 1998.

[7] Alessia Bardi, Paolo Manghi, and Franco Zoppi. Aggregative Data Infrastructures for the Cultural Heritage. In *Research Conference on Metadata and Semantic Research*, pages 239–251, 2012.

[8] Roger Barga, Jared Jackson, Nelson Araujo, Dean Guo, Nitin Gautam, and Yogesh Simmhan. The trident scientific workflow workbench. In *Proceedings - 4th IEEE International Conference on eScience, eScience 2008*, pages 317–318, 2008.

[9] Adam Barker and Jano Van Hemert. Scientific Workflow: A Survey and Research Directions. In *International Conference on Parallel Processing and Applied Mathematics*, pages 746–753, 2007.

[10] Carlo Batini, Daniele Barone, Federico Cabitza, and Simone Grega. A Data Quality Methodology for Heterogeneous Data. *International Journal of Database Management Systems*, 3(1):60–79, 2011.

[11] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM Computing Surveys*, 41(3):16:1–16:52, 2009.

[12] Carlo Batini, Matteo Palmonari, and Gianluigi Viscusi. The Many Faces of Information and their Impact on Information Quality. In *Proceedings of the 17th International Conference on Information Quality*, pages 212–228, 2012.

[13] Carlo Batini, Anisa Rula, Monica Scannapieco, and Gianluigi Viscusi. From Data Quality to Big Data Quality. *Journal of Database Management*, 26(1):60–82, 2015.

[14] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[15] Carlo Batini and Monica Scannapieco. *Data and Information Quality*. Springer International Publishing, 2016.

[16] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[17] Helen M. Berman, Philip E. Bourne, and John Westbrook. The Protein Data Bank: A case study in management of community data. *Current Proteomics*, 1(1):49–57, 2004.

[18] Martin Bichler and Kwei-Jay Lin. Service-Oriented Computing. *Computer*, 39(3):99–101, mar 2006.

[19] Mónica Bobrowski, Martina Marré, and Daniel Yankelevich. Measuring data quality. Technical report, Universidad de Buenos Aires, Buenos Aires, 1999.

[20] Christine L. Borgman. *Big data, little data, no data: Scholarship in the networked world*. Mit Press, 2015.

[21] F. Boufares and A. Ben Salem. Heterogeneous data-integration and data quality: Overview of conflicts. *2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications, SETIT 2012*, pages 867–874, 2012.

[22] Thomas R. Bruce and Diane I. Hillmann. The Continuum of Metadata Quality: Defining, Expressing, Exploiting. In *Metadata in Practice*, pages 1–18. ALA Editions, 2004.

[23] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *International conference on database theory (ICDT)*, pages 316—-330. Springer-Verlag, 2001.

[24] Li Cai and Yangyong Zhu. The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal*, 14:2, 2015.

[25] Leonardo Candela, Donatella Castelli, and Pasquale Pagano. Virtual Research Environments: An Overview and a Research Agenda. *Data Science Journal*, 12(August):GRDI75–GRDI81, 2013.

[26] Jorge Cardoso, Amit Sheth, and John Miller. Workflow quality of service. In *IFIP Advances in Information and Communication Technology*, volume 108, pages 303–311, 2003.

[27] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics*, 1(3):281–308, 2004.

[28] Marcus Christie and Suresh Marru. The LEAD Portal: a TeraGrid gateway and application service architecture. *Concurrency and Computation: Practice and Experience*, 19(6):767–781, apr 2007.

[29] Susan B. Davidson, Sarah Cohen Boulakia, Anat Eyal, Bertram Ludäscher, Timothy M. McPhillips, Shawn Bowers, Manish Kumar Anand, and Juliana Freire. Provenance in Scientific Workflow Systems. *Data Engineering Bulletin*, 30(4):1–7, 2007.

[30] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping Scientific Workflows onto the Grid. *Grid Computing*, pages 11–20, 2004.

[31] Rion Dooley, Kent Milfeld, Chona Guiang, Sudhakar Pamidighantam, and Gabrielle Allen. From proposal to production: Lessons learned developing the computational chemistry Grid cyberinfrastructure. *Journal of Grid Computing*, 4(2):195–208, 2006.

[32] Sharanya Eswaran, David Vecchio, Glenn Wasson, and Marty Humphrey. Adapting and Evaluating Commercial Workflow Engines for e-Science. In *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, pages 20–20. IEEE, dec 2006.

[33] Lorena Etcheverry, Verónika Peralta, and Mokrane Bouzeghoub. Qbox-foundation: a metadata platform for quality measurement. In *Proceeding of the 4th Workshop on Data and Knowledge Quality*, pages 1–10, 2008.

[34] Thomas Fahringer, Radu Prodan, Rubing Duan, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong Linh Truong, Alex Villazón, and Marek Wieczorek. ASKALON: A grid application development and computing environment. In *Proceedings - IEEE/ACM International Workshop on Grid Computing*, volume 2005, pages 122–131, 2005.

[35] Lemos Fernando, Mohamed Reda Bouadjenek, Mokrane Bouzeghoub, and Zoubida Kedad. Using the QBox platform to assess quality in data integration systems. *Ingenierie des systemes d'information*, 15(6):105–124, 2010.

[36] Craig W. Fisher and Bruce R. Kingma. Criticality of data quality as exemplified in two disasters. *Information and Management*, 39(2):109–116, 2001.

[37] Jennifer J. Gassman, Walter W. Owen, Timothy E. Kuntz, Jeffrey P. Martin, and William P. Amoroso. Data quality assurance, monitoring, and reporting. *Controlled Clinical Trials*, 16(2):104–136, 1995.

[38] Jonathan J Geiger. Data quality management: the most critical initiative you can implement. In *SUGI 29 Proceedings*, pages 1–14, 2004.

[39] Michael Gertz, M. Tamer Özsu, Gunter Saake, and Kai-Uwe Sattler. Report on the Dagstuhl Seminar "Data Quality on the Web" Results from Working Groups Metadata and Modeling. *SIGMOD Record*, 33(1):127–132, 2004.

[40] Belinda Giardine, Cathy Riemer, Ross C. Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, Webb Miller, W. James Kent, and Anton Nekrutenko. Galaxy: A platform for interactive large-scale genome analysis. *Genome Research*, 15(10):1451–1455, 2005.

[41] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, 2007.

[42] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86, 2010.

[43] Laura González, Verónika Peralta, Mokrane Bouzeghoub, and Raúl Ruggia. Qbox-services: Towards a service-oriented quality platform. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5833 LNCS, pages 232–242, 2009.

[44] Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova, Frank Leymann, and Michael Reiter. *Conventional Simulation Workflow Technology for Scientific Conventional Workflow Technology for Scientific Simulation*. Springer-Verlag, 2011.

[45] Jim Gray. Jim Gray on eScience: A transformed scientific method. In *The fourth paradigm: data-intensive scientific discovery*, pages 17–31. Microsoft Research, 2009.

[46] Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. Scientific data management in the coming decade. *SIGMOD Rec.*, 34(4):34–41, 2005.

[47] Dan Gunter, Ewa Deelman, Taghrid Samak, Christopher H. Brooks, Monte Goode, Gideon Juve, Gaurang Mehta, Priscilla Moraes, Fabio Silva, Martin Swany, and Karan Vahi. Online workflow management and performance analysis with Stampede. In *7th International Conference on Network and Service Management, CNSM 2011*, pages 1–10, 2011.

[48] Tony Hey, Stewart Tansley, and Kristin Tolle. *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research, 2009.

[49] Y.U. Huh, F.R. Keller, Thomas C. Redman, and A.R. Watkins. Data quality. *Information and Software Technology*, 32(8):559–565, 1990.

[50] Gerhard Klimeck, Michael McLennan, Sean P Brophy, George B Adams, and Mark S Lundstrom. nanohub. org: Advancing education and research in nanotechnology. *Computing in Science & Engineering*, 10(5):17–23, 2008.

[51] Petr Knoth and Zdenek Zdrahal. CORE: Three access levels to underpin open access. *D-Lib Magazine*, 18(11-12), 2012.

[52] Mateusz Kobos, Lukasz Bolikowski, Marek Horst, Paolo Manghi, Natalia Manola, and Jochen Schirrwagen. Information inference in scholarly communication infrastructures: the OpenAIREplus project experience. In *Proceeedings of the 10th Italian Research Conference on Digital Libraries*, volume 38, pages 92–99, 2014.

[53] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing*, 13(4):457–493, 2015.

[54] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[55] Joana E Gonzales Malaverri, André Santanchè, and Claudia Bauzer Medeiros. A provenance–based approach to evaluate data quality in eScience. *International Journal of Metadata, Semantics and Ontologies*, 9(1):15–28, 2014.

[56] Joana Esther Gonzales Malaverri. *Supporting data quality assessment in eScience: a provenance based approach*. PhD thesis, University of Campinas, 2013.

[57] Paolo Manghi, Michele Artini, Claudio Atzori, Alessia Bardi, Andrea Mannocci, Sandro La Bruzzo, Leonardo Candela, Donatella Castelli, and Pasquale Pagano. The D-NET software toolkit. *Program*, 48(4):322–354, 2014.

[58] Paolo Manghi, Lukasz Bolikowski, Natalia Manola, Jochen Schirrwagen, and Tim Smith. OpenAIREplus: The European scholarly communication data infrastructure. *D-Lib Magazine*, 18(9-10), 2012.

[59] Paolo Manghi, Natalia Manola, Wolfram Horstmann, and Dale Peters. An Infrastructure for Managing EC Funded Research Output-The OpenAIRE Project. *International Journal on Grey Literature*, 6:31–40, 2010.

[60] Andrea Mannocci, Vittore Casarosa, Paolo Manghi, and Franco Zoppi. The Europeana Network of Ancient Greek and Latin Epigraphy Data Infrastructure. In *Research Conference on Metadata and Semantics Research*, pages 286–300. Springer International Publishing, 2014.

[61] Andrea Mannocci and Paolo Manghi. DataQ: A Data Flow Quality Monitoring System for Aggregative Data Infrastructures. In *International Conference on Theory and Practice of Digital Libraries*, pages 357–369. Springer International Publishing, 2016.

[62] Thomas Margaritopoulos, Merkourios Margaritopoulos, Ioannis Mavridis, and Athanasios Manitsaris. A conceptual framework for metadata quality assessment. In *Proceedings of the 2008 International Conference on Dublin Core and Metadata Applications*, pages 104–113, 2008.

[63] Adriana Marotta, Laura González, and Raúl Ruggia. A Quality Aware Service-oriented Web Warehouse Platform. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, volume 565, pages 29–32, 2012.

[64] Adriana Marotta and Raúl Ruggia. Quality Management in Multi-Source Information Systems. *Quality*, 2002.

[65] Adriana Marotta and Raúl Ruggia. Managing source quality changes in a data integration system. In *CEUR Workshop Proceedings*, volume 263, pages 1168–1176, 2006.

[66] Marta Mattoso, Jonas Dias, Kary A C S Ocaña, Eduardo Ogasawara, Flavio Costa, Felipe Horta, Vítor Silva, and Daniel De Oliveira. Dynamic steering of HPC scientific workflows: A survey. *Future Generation Computer Systems*, 46:100–113, 2015.

[67] Michael McLennan and Rick Kennell. HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering. *Computing in Science & Engineering*, 12(2):48–53, mar 2010.

[68] Massimo Mecella, Monica Scannapieco, Antonino Virgillito, Roberto Baldoni, Tiziana Catarci, and Carlo Batini. Managing Data Quality in Cooperative Information Systems. *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, 2519:486–502, 2002.

[69] Paolo Missier. *Modelling and Computing the Quality of Information in E-Science*. PhD thesis, University of Manchester, 2008.

[70] Paolo Missier, Suzanne Embury, Mark Greenwood, Alun Preece, and Binling Jin. Quality Views: Capturing and Exploiting the User Perspective on Data Quality. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, pages 977–988, 2006.

[71] Paolo Missier, Suzanne M. Embury, Mark Greenwood, Alun Preece, and Binling Jin. Managing information quality in e-science. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, page 1150, New York, New York, USA, 2007. ACM Press.

[72] Paolo Missier, Alun Preece, Suzanne Embury, Binling Jin, Mark Greenwood, David Stead, and Al Brown. Managing information quality in e-Science: A case study in proteomics. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3770:423–432, 2005.

[73] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, and Others. The open provenance model core specification (v1. 1). *Future generation computer systems*, 27(6):743–756, 2011.

[74] Aisa Na'im, Daniel Crawl, Maria Indrawan, Ilkay Altintas, and Shulei Sun. Monitoring Data Quality in Kepler. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 560–564, 2010.

[75] Felix Naumann, U. Leser, and J.C. Freytag. Quality-driven integration of heterogeneous information systems. In *Proceedings of the International Conference on Very Large Data Bases*, pages 447–458, 1999.

[76] Hoang Nguyen and David Abramson. WorkWays: Interactive workflow-based science gateways. *8th International Conference on E-Science*, 2012.

[77] Eduardo Ogasawara, Jonas Dias, Vítor Silva, Fernando Chirigati, Daniel De Oliveira, Fabio Porto, Patrick Valduriez, and Marta Mattoso. Chiron: A parallel engine for algebraic scientific workflows. *Concurrency Computation Practice and Experience*, 25(16):2327–2341, 2013.

[78] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[79] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings - 4th International Conference on Web Information Systems Engineering, WISE 2003*, pages 3–12, 2003.

[80] Jung-Ran Park. Metadata Quality in Digital Repositories: A Survey of the Current State of the Art. *Cataloging & Classification Quarterly*, 47(3-4):213–228, 2009.

[81] Verónika Peralta. Data Freshness and Data Accuracy : A State of the Art. Technical report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, URUGUAY, 2006.

[82] Verónika Peralta. *Data Quality Evaluation in Data Integration Systems*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2006.

[83] Verónika Peralta, Raúl Ruggia, Zoubida Kedad, and Mokrane Bouzeghoub. A Framework for Data Quality Evaluation in a Data Integration System. In *SBBD*, pages 134–147, 2004.

[84] Silvio Peroni, Francesca Tomasi, and Fabio Vitali. The aggregation of heterogeneous metadata in web-based cultural heritage collections: A case study. *International Journal of Web Engineering and Technology*, 8(4):412–432, 2013.

[85] Leo L. Pipino, Yang W. Lee, and Richard Y. Wang. Data quality assessment. *Communications of the ACM*, 45(4):211, 2002.

[86] Alun Preece, Binling Jin, Edoardo Pignotti, Paolo Missier, Suzanne M. Embury, David Stead, and Al Brown. Managing Information Quality in e-Science Using Semantic Web Technology. *Procs. ESWC*, pages 472–486, 2006.

[87] Thomas C. Redman. *Data Quality for the Information Age*. Artech House, Inc., Norwood, MA, USA, 1st edition, 1997.

[88] Thomas C. Redman. The Impact of Poor Data Quality on the Typical Enterprise. *Communications of the ACM*, 41(2):79–82, 1998.

[89] Michael Reiter, Uwe Breitenbücher, Schahram Dustdar, Dimka Karastoyanova, Frank Leymann, and Hong-Linh Truong. A novel framework for monitoring and analyzing quality of data in simulation workflows. In *Proceedings of the 7th International Conference on e-Science*, pages 105–112. IEEE, 2011.

[90] Michael Reiter, Uwe Breitenbucher, Oliver Kopp, and Dimka Karastoyanova. Quality of data driven simulation workflows. In *2012 IEEE 8th International Conference on E-Science*, pages 1–8. IEEE, 2012.

[91] Michael Reiter, Uwe Breitenbücher, Oliver Kopp, and Dimka Karastoyanova. Quality of data driven simulation workflows. *Journal of Systems Integration*, 5(1):3–29, 2014.

[92] Barna Saha and Divesh Srivastava. Data quality: The other face of Big Data. In *2014 IEEE 30th International Conference on Data Engineering*, pages 1294–1297. IEEE, mar 2014.

[93] Monica Scannapieco, Paolo Missier, and Carlo Batini. Data Quality at a Glance. *Datenbank-Spektrum*, 14(January):6–14, 2005.

[94] Jochen Schirrwagen, Paolo Manghi, Natalia Manola, Lukasz Bolikowski, Najla Rettberg, and Birgit Schmidt. Data curation in the openaire scholarly communication infrastructure. *Information Standards Quarterly*, 25(3):13—-19, 2013.

[95] G. Shankaranarayanan, Richard Y. Wang, and M. Ziad. IP-MAP: Representing the Manufacture of an Information Product. In *Proceedings of the 2000 Conference on Information Quality*, pages 1–16, 2000.

[96] Sarah L. Shreeves, Ellen M. Knutson, Besiki Stvilia, Carole L. Palmer, Michael B. Twidale, and Timothy W. Cole. Is 'quality' metadata 'shareable' metadata? The implications of local metadata practices for federated collections. In *Proceedings of the 12th National Conference on Association of College and Research Libraries*, pages 223–237, 2005.

[97] Vítor Silva, Daniel de Oliveira, and Marta Mattoso. Exploratory Analysis of Raw Data Files through Dataflows. In *Proceedings of the International Symposium on Computer Architecture and High Performance Computing Workshop*, pages 114–119, 2014.

[98] Vítor Silva, Daniel de Oliveira, Patrick Valduriez, and Marta Mattoso. Analyzing related raw data files through dataflows. *Concurrency and Computation: Practice and Experience*, 2015.

[99] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Survey of Data Provenance in e-Science, 2005.

[100] J. Sivic and A. Zisserman. Video Google: a text retrieval approach to object matching in videos. In *Proceedings of the 9th International Conference on Computer Vision*, pages 1470–1477, 2003.

[101] Renato Beserra Sousa, Daniel Cintra Cugler, Joana Esther, Gonzales Malaverri, and Claudia Bauzer Medeiros. A provenance-based approach to manage long term preservation of scientific data. In *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*, pages 126–133. IEEE, 2014.

[102] Diane M. Strong, Yang W. Lee, and Richard Y. Wang. Data quality in context. *Communications of the ACM*, 40(5):103–110, 1997.

[103] Besiki Stvilia, Les Gasser, Michael B. Twidale, Sarah L. Shreeves, and Timothy W. Cole. Metadata quality for federated collections. In *Proceedings of the 9th International Conference on Information Quality*, pages 111–125, 2004.

[104] Alice Tani, Leonardo Candela, and Donatella Castelli. Dealing with metadata quality: The legacy of digital library efforts. *Information Processing and Management*, 49(6):1194–1205, 2013.

[105] Giri Kumar Tayi and Donald P. Ballou. Examining data quality. *Communications of the ACM*, 41(2):54–57, 1998.

[106] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The Triana Workflow Environment: Architecture and Applications. In *Workflows for e-Science*, pages 320–339. Springer London, London, 2007.

[107] Costantino Thanos. A Vision for Open Cyber-Scholarly Infrastructures. *Publications*, 4(2):13, may 2016.

[108] Ion-george Todoran, Laurent Lecornu, Ali Khenchaf, and Jean-Marc Le Caillec. A Methodology to Evaluate Important Dimensions of Information. *ACM Journal of Data and Information Quality*, 6(2 - 3, Article 11):23, 2015.

[109] Hong Linh Truong and Schahram Dustdar. On evaluating and publishing data concerns for data as a service. In *Proceedings of the 2010 IEEE Asia-Pacific Services Computing Conference*, pages 363–370, 2010.

[110] James Turnbull. *The art of monitoring*. James Turnbull, 2016.

[111] Lassi Tuura, A Meyer, I Segoni, and G Della Ricca. CMS data quality monitoring: systems and experiences. *Journal of Physics: Conference Series*, 219(7), 2010.

[112] Karan Vahi, Ian Harvey, Taghrid Samak, Daniel Gunter, Kieran Evans, Dave Rogers, Ian Taylor, Monte Goode, Fabio Silva, Eddie Al-Shkarchi, Gaurang Mehta, Andrew Jones, and Ewa Deelman. A General approach to real-time workflow monitoring. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 108–118, 2012.

[113] María Carolina Valverde, Diego Vallespir, Adriana Marotta, and Jose Ignacio Panach. Applying a data quality model to experiments in software engineering. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8823:168–177, 2014.

[114] Jillian C. Wallis, Elizabeth Rolando, and Christine L. Borgman. If We Share Data, Will Anyone Use Them? Data Sharing and Reuse in the Long Tail of Science and Technology. *PLoS ONE*, 8(7), 2013.

[115] Yair Wand and Richard Y. Wang. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11):86–95, 1996.

[116] Jianing Wang. A Quality Framework for Data Integration. In *Data Security and Security Data. 27th British National Conference on Databases, BNCOD 27. Revised Selected Papers*, pages 131–134. Springer, 2012.

[117] Richard Y. Wang, Henry B. Kon, and Stuart E. Madnick. Data quality requirements analysis and modeling. In *Proceedings of IEEE 9th International Conference on Data Engineering*, pages 670–677. IEEE Comput. Soc. Press, 1993.

[118] Richard Y. Wang, Martin P. Reddy, and Henry B. Kon. Toward quality data: An attribute-based approach. *Decision Support Systems*, 13(3-4):349–372, 1995.

[119] Richard Y. Wang, Storey, and C.P. Firth. A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):623–640, 1995.

[120] Richard Y. Wang and Diane M. Strong. Beyond Accuracy : What Data Quality Means to Data Consumers. *Management Information Systems*, 12(4):5–34, 1996.

[121] Nancy Wilkins-Diehr, Dennis Gannon, Gerhard Klimeck, Scott Oster, and Sudhakar Pamidighantam. TeraGrid Science Gateways and Their Impact on Science. *Computer*, 41(11):32–41, nov 2008.