SCIENTIFIC WORKFLOWS WITH SUPPORT OF KNOWLEDGE BASES

Victor Soares Bursztyn

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Jonas Furtado Dias

Rio de Janeiro
Agosto de 2017

SCIENTIFIC WORKFLOWS WITH SUPPORT OF KNOWLEDGE BASES

Victor Soares Bursztyn

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof[a]. Marta Lima de Queirós Mattoso, D.Sc.

_____

Prof. Fabio Andre Machado Porto, D.Sc.

_____

Prof. Alexandre de Assis Bento Lima, D.Sc.

_____

Prof[a]. Kary Ann del Carmen Ocaña, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

AGOSTO DE 2017

Bursztyn, Victor Soares

Scientific Workflows with Support of Knowledge Bases/ Victor Soares Bursztyn. – Rio de Janeiro: UFRJ/COPPE, 2017.

XII, 62 p.: il.; 29,7 cm.

Orientadores: Marta Lima de Queirós Mattoso

Jonas Furtado Dias

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2017.

Referências Bibliográficas: p. 58-62.

1. Workflows Científicos. 2. Bases de Conhecimento. 3. Human-in-the-Loop. I. Mattoso, Marta Lima de Queirós *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III Título.

À Vanessa, à Regina e ao Luiz.

AGRADECIMENTOS

Mesmo antes de frequentá-la, a UFRJ sempre foi, para mim, muito mais que um espaço de aprendizado. Antes de tudo, é um espaço de afeto, gratidão e de esperança, por sua capacidade maravilhosa de mudar vidas. Ao lado da UFRJ, como instituição, agradeço ao meu avô, Pedro, e ao meu pai, Luiz, por terem investido nessa direção e por abrirem o caminho que estou trilhando. Vovô Pedro não pôde cursar a engenharia, que era o seu sonho, mas entregou os três filhos à UFRJ. Agradeço a ele por fazer da educação a nossa estrela-guia e pela permanente lembrança de que devo ser grato às oportunidades que hoje estão ao meu alcance.

Ao meu pai, Luiz, agradeço por sempre acompanhar, com carinho e interesse, todos os passos dados até aqui, das brincadeiras de marcenaria ao primeiro livro de Turbo Pascal na sétima série. À Regina, minha mãe, agradeço por toda a energia criativa, inesgotável e contagiante. Muito obrigado pelo zelo, pelo apoio e por serem, mãe e pai, os meus maiores professores. Aos meus irmãos, Henrique e Ivan, agradeço pelas referências ao longo de suas trajetórias neste mesmo Centro de Tecnologia (e além). À Vanessa, minha esposa, agradeço por cada instante em que sonhamos juntos. Muito obrigado por nosso amor, que traz a paz para enfrentar tantos desafios. Aproveito para agradecer a toda a família, incluindo Luiz Sérgio, Suely e nossa querida Tereza.

Registro um agradecimento especial à professora Marta Mattoso e ao Jonas Dias, orientadora e co-orientador deste trabalho. Agradeço à professora por sua orientação, sempre cuidadosa e didática, e pelo apoio dado para que eu perseguisse todos os meus objetivos. Agradeço ao Jonas por ser um verdadeiro exemplo de profissional e de pessoa, alguém que inspira pela forma como pensa e age. Levo os seus exemplos para a vida. Faço outro agradecimento especial à Kary Ocaña, nossa super especialista de domínio, fundamental para o aprofundamento de nossas ideias. Ao lado da Kary, agradeço aos professores Fábio Porto e Alexandre de Assis por aceitarem fazer parte desta banca. Registro um agradecimento ao Vítor Silva, um amigo de quem pude me aproximar academicamente. Agradeço ao professor Daniel Figueiredo e ao Marcelo Granja pelo *Best Paper* no BraSNAM 2016, uma experiência acadêmica complementar a este trabalho e muito prazerosa. Agradeço aos demais professores e a toda a equipe do PESC. Por fim, agradeço ao professor e amigo Alexandre Evsukoff, do PEC, pela iniciação à pesquisa ainda na graduação.

Não poderia deixar de agradecer, também, ao Centro de P&D da Dell EMC. Pela maior parte deste mestrado, tive a insubstituível oportunidade de trabalhar e aprender

com um grupo incrível de pesquisadores. Agradeço à Karin, ao Fred, à Ana, ao Angelo, ao Chung, à Bruna e à Taíza; agradeço à Adriana, ao André, ao Caio, ao Calmon, ao Diego, ao Edward, ao Jaumir, novamente ao Jonas, à Márcia, ao Percy, ao Rômulo, ao Senra e ao Vinícius. Ainda tenho dificuldades para assimilar a extensão do que aprendi com todos vocês no Parque Tecnológico da UFRJ. Agradeço a toda a equipe da *Legacy EMC* por essa experiência valiosíssima.

Por fim, agradeço à CAPES pela bolsa ao longo dos três primeiros trimestres de mestrado, retornada espontaneamente a partir do primeiro mês no Centro de P&D. Faço votos para que o poder público cuide da UFRJ com o respeito que este espaço transformador merece.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

WORKFLOWS CIENTÍFICOS COM APOIO DE BASES DE CONHECIMENTO

Victor Soares Bursztyn

Agosto/2017

Orientadores: Marta Lima de Queirós Mattoso

Jonas Furtado Dias

Programa: Engenharia de Sistemas e Computação

Como ancorar os dados no conhecimento de domínio é um desafio frequente em experimentos de larga escala. Por um lado, a responsabilidade é inerente ao papel dos especialistas de domínio. Por outro, a larga escala de fatos de domínio, aliada à crescente complexidade, tornam esse papel trabalhoso, suscetível a erros e certas vezes inviável. Existem, entretanto, ferramentas computacionais que poderiam ajudar a lidar com essas dificuldades, melhorando as condições para pesquisas científicas baseadas em dados.

Nós estudamos esse desafio e as ferramentas existentes para propor uma abordagem que permita ancorar dados experimentais nas fases de composição, execução e análise do ciclo de vida de experimentos científicos. Para tal, projetamos dois experimentos: o primeiro cobrindo a fase de análise e o segundo cobrindo a fase de composição. No primeiro experimento, recorremos ao estado-da-arte para construção de bases de conhecimento a fim de organizar um conhecimento de domínio que se encontra espalhado por fontes de dados heterogêneas. No segundo, aproveitamos o estado-da-arte em computação interativa a fim de absorver o conhecimento de bases já estabelecidas, disponíveis pela Internet. Em ambos, discutimos como tais ferramentas podem levar conhecimento relevante ao *loop* de experimentos científicos, apoiando o *human-in-the-loop* (HIL). Os resultados experimentais mostram que nossa abordagem pode viabilizar experimentos que seriam difíceis ou impossíveis com o HIL tradicional. Por fim, discutimos como motores para execução de workflows e seus dados de proveniência poderiam ser usados a fim de estender este trabalho à fase de execução de experimentos científicos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SCIENTIFIC WORKFLOWS WITH SUPPORT OF KNOWLEDGE BASES

Victor Soares Bursztyn

August/2017

Advisors: Marta Lima de Queirós Mattoso

Jonas Furtado Dias

Department: Computer Science Engineering

Finding the best way to make the data well-grounded in domain knowledge is an important challenge in large-scale experiments. While this responsibility inherently depends on the role of domain experts, the large amount of domain-related facts and their associated complexity increasingly makes this role too labor-intensive, susceptible to errors, and sometimes unfeasible. However, there are computational tools that could help to cope with these difficulties, thus enhancing the conditions for data-driven science.

We study the aforementioned challenge and propose a set of tools to help grounding the data in the composition, execution and analysis phases of the scientific experiment lifecycle. We design two experiments: the first focusing on the analysis phase and the second with focus on the composition phase. In the first, we resort to the current state-of-the-art technology in knowledge base construction in order to organize domain knowledge scattered across heterogeneous data sources. In the second, we leverage the state-of-the-art environment for interactive computing in order to tap into well-established knowledge bases. In both experiments we discuss how such technologies can bring relevant knowledge to the loop of scientific experiments, approaching human-in-the-loop support (HIL). The obtained experimental results show that our approach may enable a breed of experiments that could be unfeasible with traditional HIL. Finally, we discuss how provenance data could be binded to knowledge bases and leveraged by workflow engines to enable further research on the execution phase.

Índice

Listagem de Figuras

Listagem de Tabelas

Chapter 1 – Introduction

Both scientific and corporate worlds have seen the rise of data-intensive applications, based on an increased accessibility of High-Performance Computing (HPC) resources. In the scientific domains, HPC has been key to the development of a category of experiments called *in silico* experiments (Travassos and Barros 2003), which refers to applications where both the subjects of study and their environments are computationally created. In this context, workflow modeling rises as a particularly suitable way to capture the intricate sequence of steps in such applications. When planning and managing scientific experiment execution in HPC, workflow-related concepts often serve as common ground between domain experts and computer scientists (collectively referred to as "scientists"), enabling multidisciplinary research endeavors. However, multidisciplinary groups still cope with a broad set of open challenges involving, more specifically, putting domain experts in the loop of the experiment. In this sense, human-in-the-loop (HIL) (Bronaugh 2007) is a relevant and up-to-date research topic addressed by the workflow community in data-intensive applications (Jagadish *et al.* 2014).

Previous works (Mattoso *et al.* 2015) have been approaching this research topic from the data provenance standpoint (Davidson and Freire 2008). More particularly, data provenance is studied by means of the different types of challenges that come from monitoring data products in intermediate steps of a workflow: from data extraction to making sense of it (e.g., information retrieval, data visualization, among other higher-level data manipulations), eventually including the capacity to act on an insight at the most opportunistic time (Dias *et al.* 2015). These works have been conducted in partnership with real-world domain experts, who ultimately benefit from it, through Chiron, a workflow execution engine (Ogasawara *et al.* 2013). Chiron's major goals are: to make effective use of HPC resources; to enable insightful data provenance at runtime; and to support dynamic steering, so that long-lasting workflows can be actively managed by domain experts.

Chiron addresses the challenges related to HIL based on a well-defined lifecycle for scientific experiments (Mattoso *et al.* 2010). It consists of three major phases: first,

the composition phase; second, the execution phase; and third, the analysis phase. In Figure 1, Mattoso *et al.* illustrate the lifecycle of an experiment.

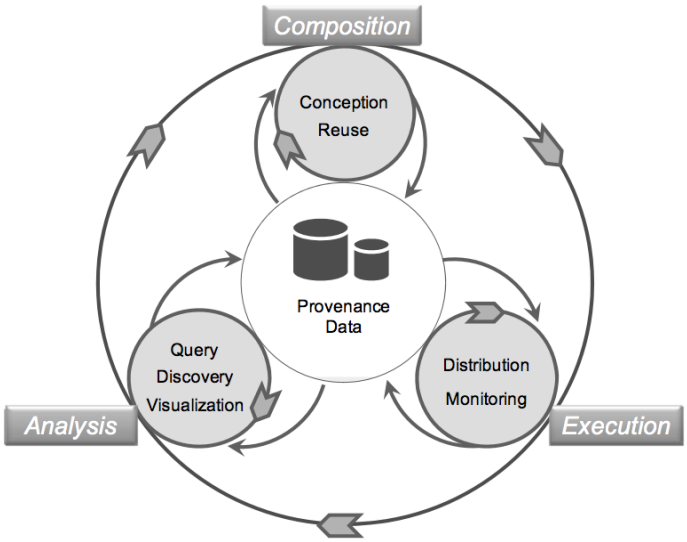

Figure 1 – The lifecycle of scientific experiments (Mattoso *et al.* 2010).

In the composition phase, scientists model the experiment by defining the chaining of activities, their parameters, the associated input data, and even the expected results. It is the phase in which scientists conceive the scientific workflow and, in the process, might opt to reutilize components or entire workflows that were previously conceived. In the execution phase, the provided workflow is instantiated in a HPC environment with the support of a given workflow execution engine. In order to perform such execution, all instances of activities are scheduled on the available computing resources following a parallel execution plan. The workflow execution engine may monitor for eventual faults (and apply suitable policies, when needed) until the workflow execution comes to its end. In the analysis phase, scientists may query and visualize the attained results in order to verify their hypotheses. They can explore the results data with the purpose of finding correlations and meaningful behaviors that lead to conclusions or to insights for further iterations, regarding either the current hypothesis or a new one. Therefore, the lifecycle could come to an end or iterate by an arbitrary number of times, depending on domain experts' criteria.

How domain experts make such decisions is a multifaceted research topic (Pirolli and Card 2005), as different aspects of this decision-making process drive largely complementary works. Hypothesis management (Gonçalves *et al.* 2014), for

instance, may address the quantitative challenges when testing competing hypotheses. Gonçalves and Porto (2015) propose a framework to compare competing theoretical models in light of their capacities to predict a common set of observations originated from a phenomenon under study. According to Gonçalves and Porto, testing competing hypotheses in a data-driven fashion is key for the development of data-driven science. In line with this goal, providing scientists with the capacity to ground their experiments in well-established domain knowledge may be another key element for supporting data-driven science.

We may formalize well-established domain knowledge as a set of well-established scientific facts (or simply "facts"). Facts can differ from raw observations because they are necessarily stable according to the state-of-the-art of a domain, that is, they are largely acknowledged as true within such domain. Facts thus emerge from global analyses comprising all relevant references within such domain. In contrast, raw observations taken by an isolated reference as true can be globally refuted for various reasons (e.g., due to experimental noise or due to untrustworthy methodologies).

In all of the three phases of the scientific experiment lifecycle, domain experts play the role of intermediaries for domain knowledge. Their goal is to ensure experiment's consistency by ensuring that it is well-grounded in domain-related facts. In order to do that, they engage in tasks that are very intensive in fact-checking, which can be defined as the process of interacting with domain-related references so to select knowledge that is relevant to the experiment (i.e., a potentially large set of facts) while applying a formal methodology. Such tasks often become too laborious or even unfeasible as the facts that are relevant grow in scale and complexity, which is a typical trait of data-intensive workflows. This makes fact-checking a very important problem in the context of HIL, and for this reason constitutes the focus of this work.

To address that problem, we hypothesize that presently available computational tools can help making those tasks more practical and feasible while preserving or even enhancing the recovery of all rationales behind domains experts' decisions (i.e., their formal methodologies). We can therefore devise the "Fact-Checker", a computational solution that: (i) performs fact-checking in a reliable manner regardless of the scale and complexity of domain knowledge; and (ii) preserves the formal methodology behind the

selection of facts, so that it can be presented in conjunction with all experimental results. Figure 2 conceptually maps how a Fact-Checker abstraction may relate to: (i) the selection of domain knowledge that must be brought to the loop of an experiment (defined as the *grounding* process), and (ii) the tracking of methodological decisions made in this experiment's context (defined as the *tracking* process). As scientists make new decisions, the Fact-Checker needs to retrieve more knowledge for the grounding process while keeping track of the methodology accordingly.

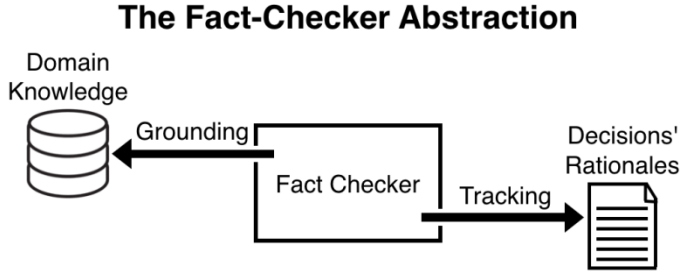

Figure 2 – The general Fact-Checker abstraction.

Additionally, it is also true that each phase requires domain experts to perform different sorts of interactions related to fact-checking. According to Endert *et al.* (2014) interaction is the critical glue that integrates analytics and human analysts, as analytic algorithms occasionally consult human experts for feedback. In this sense, we further hypothesize that each phase presents different opportunities for designing computational Fact-Checkers in order to support domain experts' interactions. We can therefore add a third statement to the Fact-Checker definition: (iii) it may be applicable to each and every phase of the lifecycle as independent solutions.

In the context of an experiment, for instance, scientists frequently need to consult domain knowledge to compose a workflow (i.e., to define its activities, parameters, and the most up-to-date input data) and to set the initial expectations. While in the execution phase, domain experts may debug the workflow through provenance data and steer it whenever they find it necessary. In this loop, as well as in the analysis phase, they often find themselves checking whether domain-related facts are being respected. If the experiment is refuting known facts, instead of validating them, it is more likely that it needs to be stopped, reconfigured, and only then restarted (after all, most of the time science extends rather than disrupts all previous knowledge). During

this process, to decide whether an experiment is in a promising direction, domain experts may resort to text notes, results from previous experiments and to domain's literature. As fact-checking in data-intensive workflows exceeds human ability to recall facts, it requires interactions with heterogeneous data sources, from structured tables to unstructured PDF files.

Parallel to the scientific workflow community, the knowledge base construction (KBC) community has been advancing towards easier to use (Gennari *et al.* 2003) and more effective systems (Niu *et al.* 2012), approaching knowledge stored in highly heterogeneous data sources. According to Niu *et al.* (2012), that is the precise definition of it: "KBC is the process of populating a knowledge base (KB) with facts (or assertions) extracted from data (e.g., text, audio, video, tables, diagrams etc)." DeepDive (Ré *et al.* 2014), a particularly mature KBC system, has assisted domain experts from different fields in their goals of structuring well-established knowledge: in paleontology and geology, in genetics and pharmacogenetics, in domains of criminal investigations, and even in the creation of an enriched version of Wikipedia (DeepDive - DeepDive Applications 2016).

To illustrate how scientific experiments could benefit from the integration of scientific workflows and KBC, consider an ecological study of population dynamics concerning various communities of wild animals in a given territory. In this setting, a group of domain experts constantly observes these communities so to track their growth. The ways by which they register observations (for simplicity, in this example, regarded as facts) are semi-structured and heterogeneous. In a computational experiment of this kind, various theoretical models could be tested by contrasting their predictions (i.e., simulated data) to the actual observations in the field (Gonçalves and Porto 2015). Therefore, if domain experts were to supervise this task (i.e., to ground the experiment), they would face several difficulties due to the scattered nature of domain knowledge. They could resort to sampling and visual inspection in order to roughly identify the most promising models. However, this qualitative effort would possibly end up being very laborious and error-prone, considering the complexity of testing competing models in a real-world setting. Even with intelligent strategies, human fact-checking is likely to fall short either in the grounding of knowledge and in the tracking

of domain experts' decisions. When it comes to the grounding of knowledge, having a KBC process could help to bring relevant knowledge to the loop of this experiment in a comprehensive and reliable manner. When it comes to the tracking of the decisions of domain experts, it is important to note that scientists could decide to run executions that disregard specific communities of wild animals (e.g., because they are noisy or less interesting). If scientists choose to remove a set of elements from a specific version of the study, the rationale of this decision should be preserved in a recoverable way. Figure 3 depicts a possible Fact-Checker for this hypothetical setting.

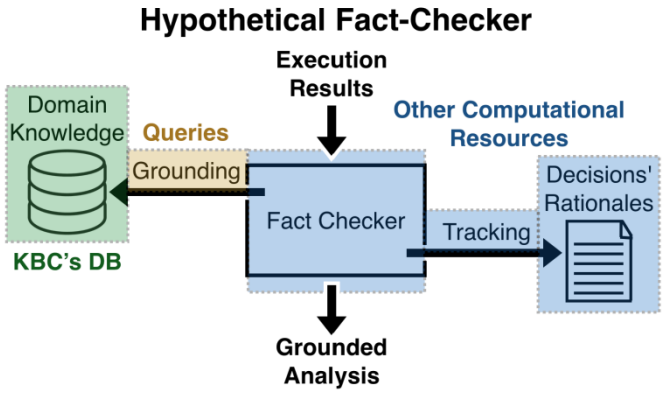

Figure 3 – A hypothetically instantiated Fact-Checker.

The motivation of this work is to enhance the conditions for data-driven science by proposing tools for grounding scientific experiments to KBs, addressing the problem of fact-checking in data-intensive workflows. We validate our hypotheses experimentally and discuss over results that we have not found in the current literature – neither in the workflow community nor in the KBC community. We highlight the advantages of binding data to scientific facts by means of scientific workflows and KBs. We consider that these KBs can be made available both locally, enabled by KBC systems such as DeepDive, or remotely, on the Internet.

In the highest level, this work pursues: how could large-scale scientific fact-checking benefit from the computational access to well-established domain knowledge?

Based on what was introduced, this question further unfolds into more tangible research topics: is it possible to build on top of the current state-of-the-art in KBC when designing a computational Fact-Checker? Is it possible to leverage well-established KBs that are made available online? With the goal of verifying our hypotheses, we propose

two use cases where we can design and evaluate computational Fact-Checkers, one in the Political Sciences domain and another in the Pharmacogenetics domain. We compare our approach to the manual fact-checking performed by domain experts in both cases. We also discuss what are the distinct opportunities in the composition phase, in the execution phase, and in the analysis phase of the scientific experiment lifecycle.

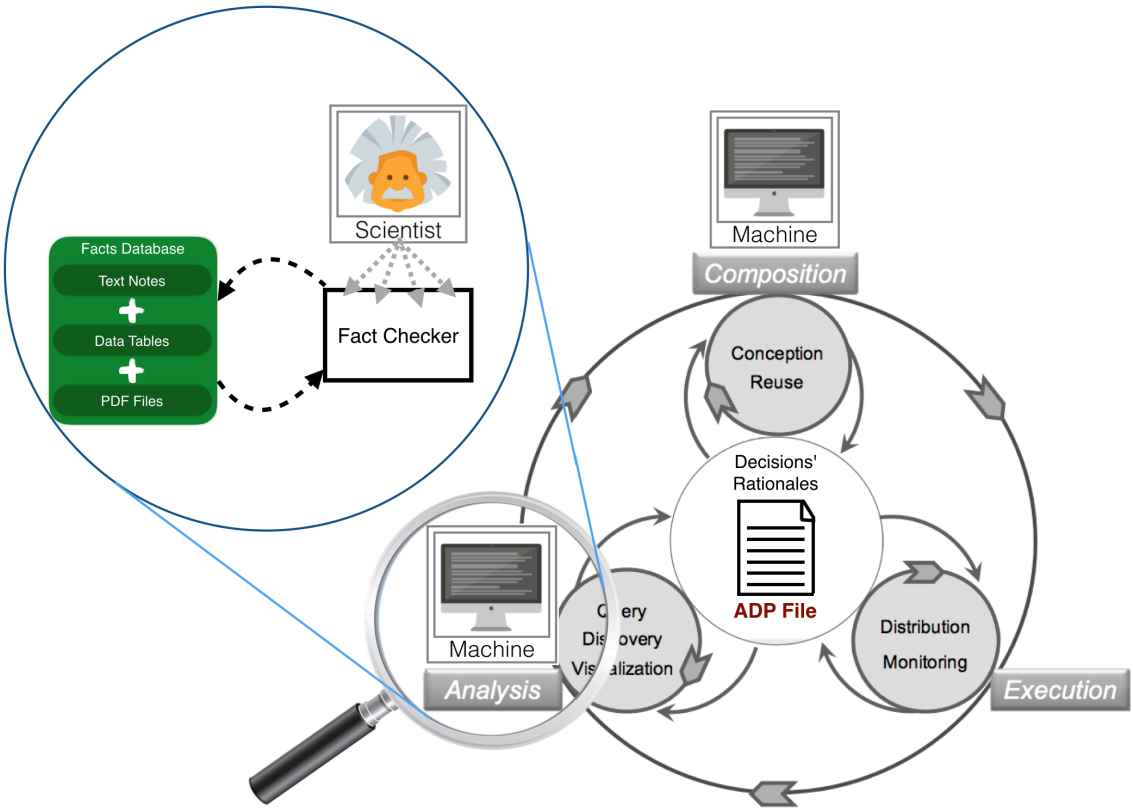

Figure 4 – Improving HIL: the Fact-Checker in the context of the scientific experiment lifecycle.

By proposing this research, we expect to improve HIL and decision processes that are strongly based on fact-checking in a given scientific domain. Figure 4 illustrates how our solution aims to support human fact-checking by undertaking repetitive human interactions in each phase of the lifecycle. It contrasts with the traditional views of the lifecycle and HIL because fact-checking starts being performed by computational means, rather than being performed by domain experts. With our Fact-Checker, machines interact with the phases of an experiment so to perform all of the necessary grounding. In turn, as domain experts are the ones who instantiate the Fact-Checker

(and further manage it), the tracking process ensures that their decisions' rationales are preserved.

With such enhancement to the HIL we aim at three desired benefits: (i) saving domain experts time in manual inspections; (ii) saving HPC resources, as it enables the workflow to converge to the right direction more quickly (in some cases, with very low dependency to human fact-checking); and (iii) in cases when experiments are deterministic, improving workflows reproducibility because even the higher level decisions can be reproduced if the original KB is provided. Such benefits may be verified or measured in the two scientific experiments devised for this research, briefly described by Table 1 and below.

Table 1 - Characteristics of Experiment 1 and Experiment 2.

	Local KB	**Remote KB**
KB Access in the Analysis Phase	Experiment 1: Clustering of Brazilian Congressmen over Voting Sections to Find Supra-Partisan Groups	
KB Access in the Composition Phase		Experiment 2: Generating Phylogenetic Trees to Study Specific Diseases

Experiment 1 consists in a workflow for clustering Brazilian congressmen according to their voting behavior in a given set of legal bills. The number of clusters is a parameter that is varied along a range of values, in order to find clusters of congressmen that accurately represent supra-partisan groups gathered around a traditional interest (agribusiness, workers' rights, religious values etc). In turn, these supra-partisan groups are described in unstructured reports, which are typically read by political scientists who are required to recall the compositions of such groups as well-established domain facts. Our results show that a KBC system can be cost-effective

when used in a Fact-Checker for the analysis phase. This approach enables large-scale fact-checking and avoids manual labor, which is time-consuming and error-prone. Furthermore, it avoids the risks that exist in alternative analytical strategies that still rely on human fact-checking (e.g., to narrow the results space by applying a heuristic), and it has the potential to improve the reproducibility of the experiment.

Experiment 2 exists in the domain of Pharmacogenetics, and consists in a workflow for studying specific diseases in light of phylogenetic trees. These trees are rendered after the processing of SciPhy, a workflow that requires as input various files representing genetic sequences. Usually, the entire decision process when selecting such input files is largely laborious and depends on a profound domain knowledge. Biologists go through all the established literature regarding a targeted disease and retrieve all genes that could be possibly related to it. Then, they would browse through online KBs dedicated to the mapping of genes from different genomes, looking for other species that are substantially related to the retrieved set of genes. Biologists may iterate on the decision process of what species to include, generating different executions of SciPhy. Our results show that online KBs can be used in the context of a Fact-Checker once we resort to specific tools to address the associated challenges. This approach shows that binding online KBs to the composition phase is a promising strategy. Compared to human fact-checking, it improves the composition phase by providing a flexible environment to analyze and retrieve domain-related facts. This environment also preserves decisions made while domain experts handle such data, allowing for better reproducibility.

This work is organized as follows: in chapter 2 we review related works that could help in the design of computational Fact-Checkers, namely scientific KBs, KBC systems (further embodied by DeepDive), and technologies to support tapping into online KBs. In chapter 3 we leverage DeepDive in order to design a Fact-Checker for the analysis phase of Experiment 1. In chapter 4 we extend existing technologies so to address previously identified research opportunities, clarifying our design choices with the support of Experiment 2. In chapter 5 we tap into a set of online KB in order to design a Fact-Checker for the composition phase of Experiment 2. At last, in chapter 6 we compare experimental evidences to human benchmarks, analyze our hypotheses in

light of such comparisons, and conclude with a list of research opportunities that arise subsequently to this work.

Chapter 2 – Related Works for Designing Fact-Checkers

In light of our Fact-Checker abstraction depicted in Figure 1, we need to review background concepts and current state-of-the-art technologies that could support the lifecycle of scientific experiments. We analyze the anatomy of KBC systems and, in particular, the anatomy of DeepDive, as this open-source system is currently the state-of-the-art in the KBC community. We also analyze how existing KBs are made available online, including the challenges and limitations involving their use. We further explain an interactive computing environment (Project Jupyter 2017) to cope with several difficulties identified when accessing online KBs. We conclude with a brief view on features that should be improved in order to address all of the important aspects in our Fact-Checker model. These missing features become research opportunities studied further on, in chapter 4.

2.1 Knowledge Bases

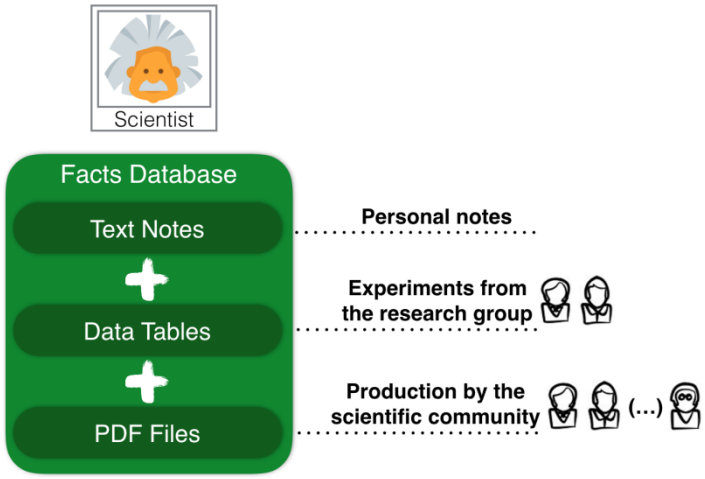

Figure 5 – What is comprised in a scientific KB.

A KB can be seen as a repository of facts (a "facts database") comprising the heterogeneous data sources from which domain experts acquire and access knowledge. Figure 5 exemplifies data sources in three levels of domain exploration: (i) the inner level represents any personal notes taken by the researcher herself; (ii) the second level represents facts discovered in partnership with a research group, often related to more

ambitious explorations that require coordination and communication of personal findings and thoughts; (iii) the outer level represents known well-established facts in that domain's scientific community. Domain expert's major challenge often lies on keeping track of well-established knowledge in its fullest extent, as it means being in line with an ever larger, globalized and productive scientific community. Therefore, the facts database should be considered an aggregation of texts, data tables and a compilation of PDF files, holding facts with different degrees of confidence – all of these aspects are topics addressed by the KBC community.

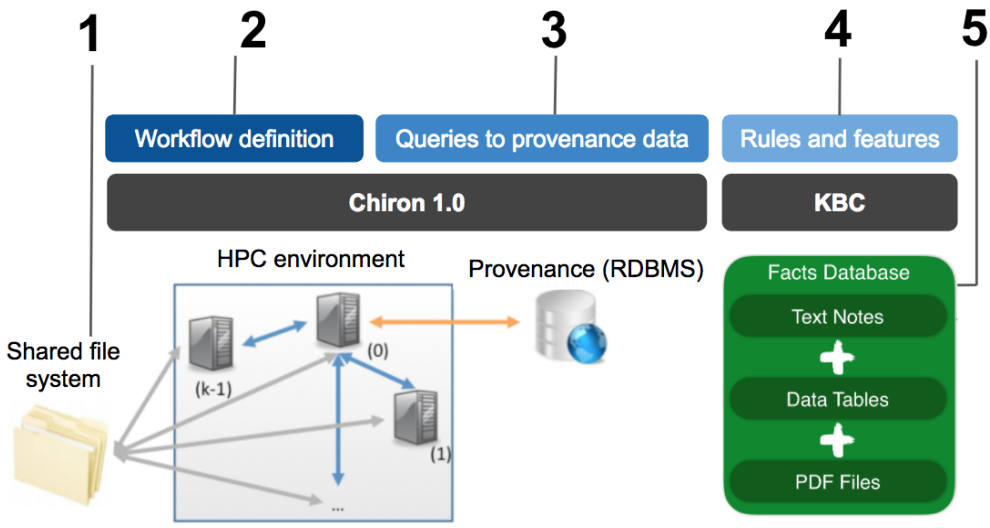

Figure 6 – Thinking the adherence of KBC technology to scientific infrastructure (Bursztyn, Dias and Mattoso 2016).

In previous works (Bursztyn, Dias and Mattoso 2016), we analyzed the adherence of KBC technology to scientific infrastructure. We discussed that a facts database could be packed along with all software artifacts defining a scientific workflow. Figure 6 shows how raw data sources and DeepDive's declarative definitions could complement the typical software artifacts found in a Scientific Workflow Management System (SWfMS), such as Chiron (Ogasawara *et al.* 2013). Chiron requires raw input data (1), a workflow definition (2), and users might run queries (3) to the provenance RDBMS. Using KBC systems would add to that architecture: the facts database (5), and definitions of features and inference rules (4). Blue boxes (2, 3, 3) represent what is expected to be tailored for the experiment. All numbered items (1, 2,

3, 4, 5) represent a fully packed experiment. Therefore, Figure 4 and 5 illustrate a basic and initial view on how to capture knowledge and incorporate it into SWfMS.

2.2 Knowledge Base Construction

According to (DeepDive - Knowledge Base Construction 2016), the KBC system's user perspective can be roughly defined in a sequence of three steps. Figure 7 sheds light on core KBC concepts involved in each step, as well as attained results, based on a partial input to the KBC system.

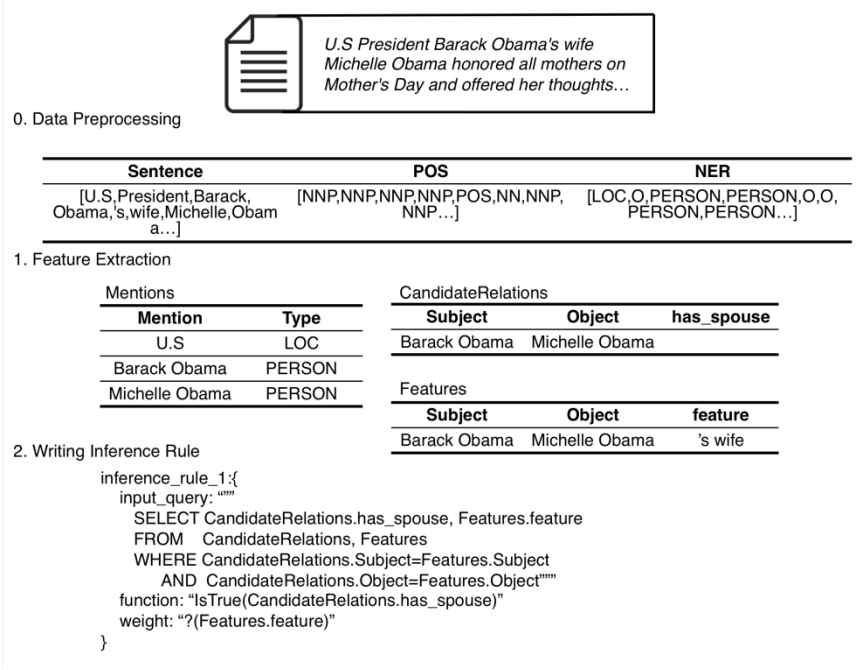

Figure 7 – How scientific KBs are constructed (DeepDive - Knowledge Base Construction 2016).

Figure 7 illustrates a KBC system aiming to populate a KB of marriages. To recall the KBC definition that was introduced previously: it illustrates a process of populating a KB with facts extracted from various data sources. In this case, former president and first lady of the US, Barack and Michelle Obama, are clearly referred as a married couple in an unstructured human-readable text input. Step 0 consists in text preprocessing: parsing sentence's words, processing part-of-speech (POS) tags and named entity tags. Step 1 is the feature extraction phase. It dives into KBC's terminology, which becomes more clear when applied further on this chapter, but may

be briefly described as follows: (i) first, it extracts *mentions* of "person" and "location" (two types of *entities* to the KBC system); (ii) then, it extracts *candidate relationships* between *entities*, of type `has_spouse`; and (iii) it also extracts *features* that potentially describe such a *relationship*, like words between *mentions* of person. Finally, Step 2 demonstrates an *inference rule*, based on data elements extracted in Step 1 and Step 0, modeled after a logic *function* and weighted a certain *factor*. *Inference rules* are the higher-level element a KBC system's user may declare before it processes *training data*, *input data* and outputs facts along with *probabilities*.

Regarding problem definition, that three-step diagram shows the user is expected to declare a set of rules and the features that are relevant to the application. In an experiment, eliciting what types of facts are relevant is a task within domain expert's capabilities.

2.3 DeepDive's Details and Specific Features

DeepDive is the current state-of-the-art for KBC (Ré *et al.* 2014). It is based on the classic Entity-Relationship (ER) model and employs popular techniques such as distant supervision and the Markov logic language, having as design goals being easy-to-use and allowing for incremental improvements over previous executions (i.e., it assumes KBC to be an iterative and live process).

The fundamental idea behind distant supervision is that large amounts of ground truths (Machine Learning supervision), even if a little noisy, are much more effective than reduced amounts of ground truths with nearly no noise. This way, DeepDive encourages having incomplete KBs as training input because they provide more examples for the learning process, which can be then applied to much larger corpora. Figure 8 illustrates the distant supervision idea in the context of a real-world complex application in its early iterations (Niu *et al.* 2012): F1-score (Figure 8's blue curve) is the harmonic average of precision (green curve) and recall (red curve), and roughly summarizes how KBC quality grows proportionally to the corpus size (Figure 8's *x*-axis), even if this means introducing more noise (i.e., more false examples).

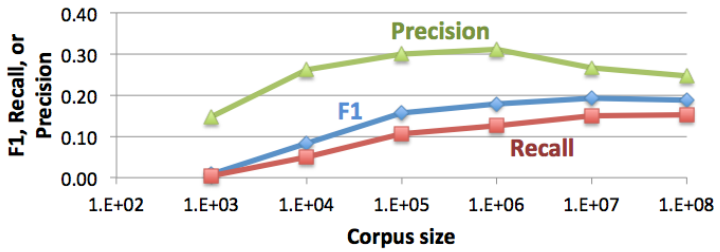

Figure 8 – The distant supervision concept is based on new insights on the trade-off between scale of Machine Learning supervision and its quality.

According to (Ré *et al.* 2014), DeepDive populates a KB first by converting raw input data into structured features using standard NLP tools and custom code. These features are then used to train statistical models representing the correlations between linguistic patterns and the targeted relationships. DeepDive combines the trained statistical models with additional knowledge into a Markov logic program that is thus used to transform the structured features (e.g., candidate entity mentions and linguistic patterns) into a KB with actual entities and relationships. All data that is structured during DeepDive's execution is stored in a RDBMS, such as EMC's Greenplum (Waas 2009).

The end-to-end process by which DeepDive transforms raw input into a comprehensive KB with entities, relationships and probabilities comprises the following set of steps:

The process of *entity linking*, which is the mapping of mentions to entities. For example, "Michelle Obama" is a straightforward mention to a real-world entity with a uniform and unique name, in this case, "Michelle_Obama_1". "Mrs. Obama" is another possible mention to the same entity "Michelle_Obama_1", although less obvious.

The process of extracting text spans (i.e., sentences) containing a pair of mentions, being a candidate relationship (i.e., a fact of interest to the KB of marriages).

The process of transforming such text spans into a set of possibly meaningful features, allowing for the identification of linguistic patterns. For example, extracting each word between the two mentions as possible features; extracting the part-of-speech (POS) tag associated to each word; or, more simply, extracting the number of words between the two mentions; among other possibilities.

The process of interpreting a declarative rule based in order to leverage knowledge that is specific to the targeted relationship. For example, a married couple necessarily refers to two distinct individuals – therefore, the KBC process may knowingly reject text spans containing pairs of mentions that refer to the same entity.

Figure 9 shows how the three steps in Figure 7 are directly mapped into DeepDive. Based on an exemplary input it depicts how a Python UDF may extract text features, which are then associated to the original input (the "MentionPairFeature" data table). Figure 9 further shows how inference rules may be declared in DeepDive using SQL, and shows how ground truths may be informed to DeepDive by means of other SQL declarations.

Figure 10 abstracts all the SQL queries to highlight two aspects of DeepDive: how incomplete KBs may be used as noisy supervision at scale (on the left of Figure 10), and how calibration plots inform users at the end of Inference and Learning cycles (on the right of Figure 10). In short, it shows that a large set of noisy examples is beneficial to DeepDive; and it shows how DeepDive is designed to consider KBC as an iterative process.

Finally, Figure 11 abstracts all the internal data tables to highlight how execution works in practice and at the high-level. It shows execution statistics for a real KB of marriages – the same example guiding all descriptions thus far. It shows the scales of the input data ("1.8M documents") and the associated output ("2.4M output facts"). It further shows that one specific Inference and Learning cycle in this iterative KBC process may be executed in 7 hours.

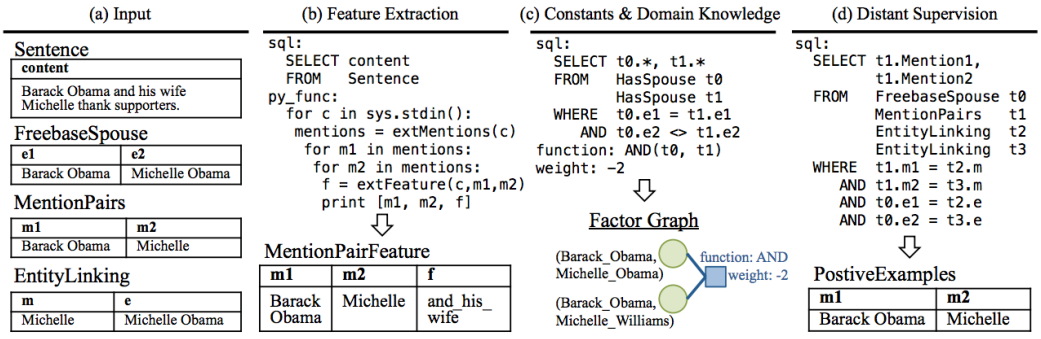

Figure 9 – Detailed process of populating a KB with DeepDive, including SQL queries and an illustrative user-defined function (UDF) in Python (Ré *et al.* 2014).

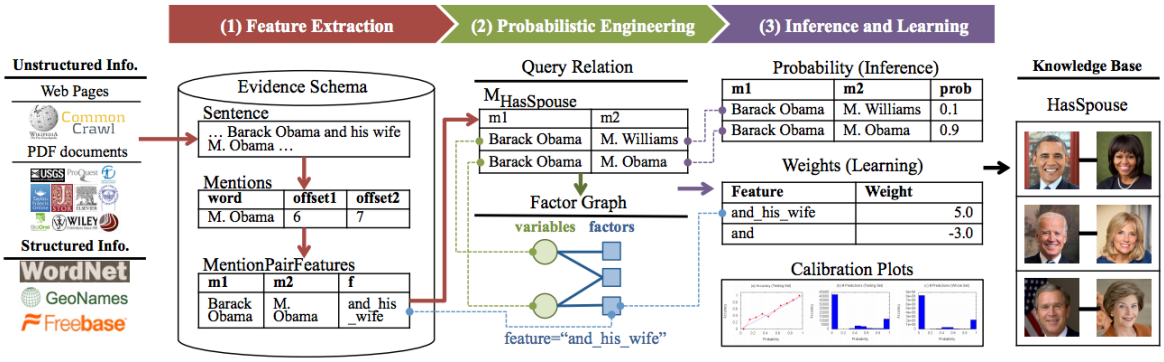

Figure 10 – Higher level iterative process of populating a KB with DeepDive (Ré *et al.* 2014).

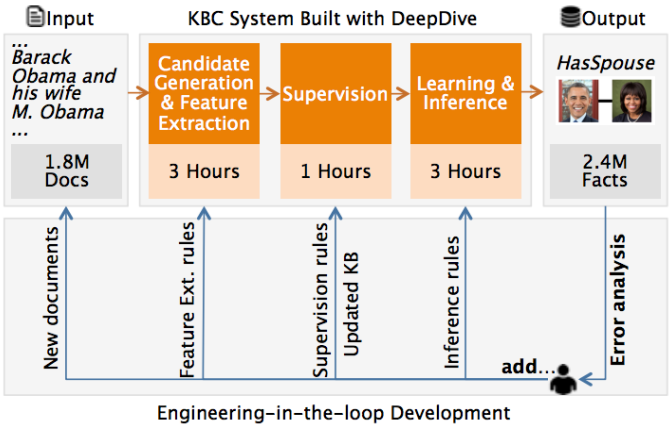

Figure 11 – DeepDive's execution details on real-world marriage KB (Shin *et al.* 2015).

2.4 Using Online Knowledge Bases

As seen in (DeepDive - DeepDive Applications 2016), several initiatives with the purpose of constructing domain-specific KBs have emerged powered by the evolution and maturation of KBC systems. Tapping into an existing online KB, however, may incur in a number of technical difficulties and open challenges.

The most fundamental interface bridging those online scientific KBs to their respective audiences (i.e., a scientific community) is what they choose to publish to a regular web browser. Through a search form, online KBs allow scientists to retrieve and visualize filtered information. As a consequence, online KBs are generally susceptible to web scraping techniques (Knox *et al.* 2007), although this is usually the most laborious way to tap into an online data source.

In order to make the effort of constructing a KB useful to a wider audience, projects resulting in online scientific KBs often comprise in its scope the creation of programmatic interfaces, that is, they publish APIs. Here, service-based APIs are in vogue due to their independency to any particular programming languages. Usually service-based APIs are designed according to RESTful principles (Masse 2011), making them not only agnostic to programming languages but also easily accessible by means of highly intelligible architectures.

More mature scientific initiatives might comprise a collection of programming language-specific libraries, such as NCBI's Entrez (Maglott 2005), providing additional benefits tied to the fact that information retrieval can be done with custom-made classes, methods or functions, which return more sophisticated data structures. For instance, in Entrez's Python library (Cock *et al.* 2009)(BioPython - Download 2017) one can associate an email to the retriever object, so that NCBI can reach out prior to canceling access rights in the event of an excessive amount of API calls.

In other cases, online scientific KBs can make their final structured data partially or entirely available for bulk download. In such situations, the dataset ends up representing a specific screenshot, with the risk of becoming outdated in live scientific initiatives that iterates over their KBs very dynamically. For this reason, bulk download is preferred in cases of more static data.

Besides the technical aspects on accessing and interacting with online scientific KBs, there are challenges of other natures: strict data access policies can impose restrictions of legal nature – usually requiring authentication, establishing an usage quota, and/or limiting data redistribution –, while HPC infrastructures can hinder the capacity to make calls to the external world. Even if those challenges are indeed of a higher order, the choice of an appropriate toolset can address and accommodate the most common constraints. In this sense, the state-of-the-art in interactive computing environments is a valuable starting point for building such a toolset.

2.5 Interactive Computing

Jupyter Notebooks (Kluyver *et al.* 2016) are an open document format based on Javascript Object Notation (JSON). The Project Jupyter claims that: "they contain a complete record of the user's sessions and embed code, narrative text, equations and rich

output" (Project Jupyter 2017) – these claims are accurate in reality and as the project's vision, but notably there are contributions yet to be made on the tracking of users' sessions (Pimentel *et al.* 2015). This part is further discussed in the next section.

The Jupyter Notebook communicates with computational Kernels using the Interactive Computing Protocol, an open network protocol based on JSON data over ZeroMQ, which is a high-performance asynchronous messaging library meant for distributed and concurrent applications, and WebSockets.

Kernels are processes that run interactive code in a particular language and return output to the user. The Jupyter project started with a support to Julia, Python and R (hence the name, Jupyter), but now the list includes over 40 supported languages. It is argued that Python, Julia and R have great appeal in scientific communities, which is a particularly good feature for the purpose of this research. Kernels also respond to tab completion and introspection requests. The Jupyter project was born out of a pioneer project of interactive kernel, the IPython project (Jupyter and the Future of IPython 2017). For this reason and because of their mutualistic relationship, both projects are often co-cited.

It supports user authentication through PAM, OAuth or by integrating with a custom directory service system. This feature makes it compliant to possible issues regarding online KBs with more restrictive access policies. Jupyter Notebooks can be deployed in a Docker container (Merkel 2014), which would typically include bulks of downloaded data accessed by the application in the Notebook. This is also a welcomed feature as it helps to address the eventual need to deploy an application by the border of HPC infrastructures with restrict access to the external world (e.g., Internet), in a format that is friendly to versioning and reproducibility (Chamberlain and Schommer 2014).

Besides allowing for programmatic explorations, which is the most important feature considering the interfaces through which online KBs are likely to open their knowledge to the world (i.e., mostly through visual interfaces plus custom-made APIs, sometimes through bulk download), another key feature is to enable domain experts to effectively interact with the integrated applications. At this point, Jupyter Notebooks become particularly suitable because of their built-in capacity to bind interactive GUI components to units of code. Figure 12 shows a real application (i.e., Experiment 2) that leverages such capacity. In this case, a search field is easily provided to domain experts

and is swiftly connected to a Python function that uses such field as input to filter data from an online KB. As Experiment 2 is presented and discussed in the chapter 5, a couple of examples of relevant interactions are also clarified, providing evidence for the importance of this feature.

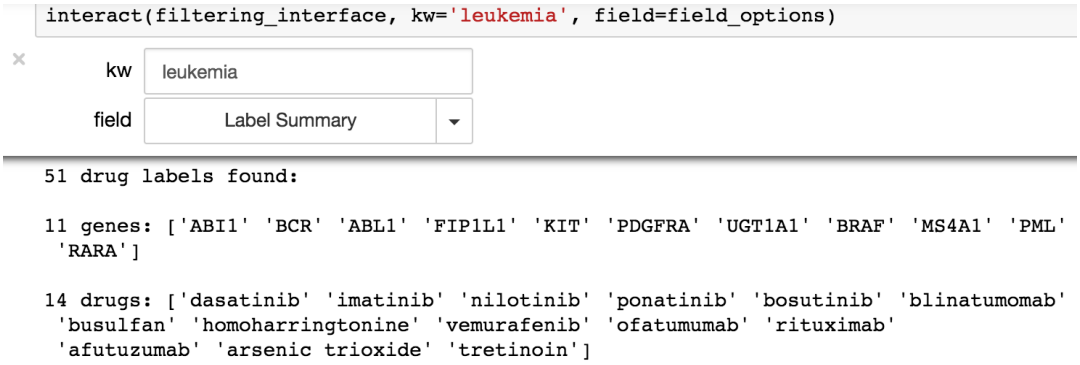

Figure 12 – Interactive GUI components come out-of-the-box and are swiftly bound to Python functions.

However, as it was recently brought for discussion by the data provenance community (Pimentel *et al.* 2015), Notebooks are falling short on their vision of recording users' sessions. Although existent, the recording of such sessions are not comprehensive: for instance, the open document format of Notebooks does not contemplate the capacity – at least not as a native feature – of keeping track of all interactions made through components. Users are left responsible for the implementation of a tracking solution, at the risk of failing to save important elements of a previously executed complex analysis. In the cases of integrated scientific solutions, more concretely, it means that higher level decisions made by scientists, or the results of a scientist's fact-checking, could all be lost.

To cope with this requirement, one alternative was recently made available and reviewed. The noWorkflow (Murta *et al.* 2014) library tracks provenance of Python scripts through reflection and other Software Engineering techniques, which makes its usage transparent and almost effortless. It allows for the exploration of previous executions from a lower level standpoint, providing the capacity to visualize function calls (as illustrated by Figure 13) and even retrieve execution details by means of SQL queries. However, tracking low level provenance data, even if effortless, could be an

overkill that adds unnecessary complexity to the needs of integrated scientific solutions at the cost of missing higher level semantics about the decision processes of domain experts.

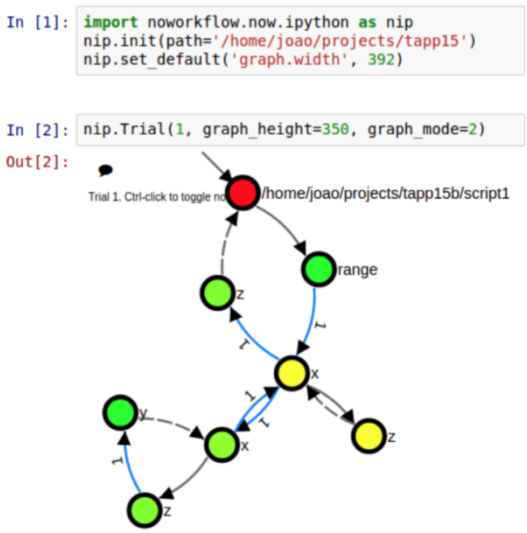

Figure 13 – Exploring the execution of a Python script using noWorkflow (Murta *et al.* 2014).

Chapter 3 – KBC to Support Fact-Checking in the Analysis Phase

To the best of our knowledge, we have not found related works with the purpose of bringing relevant knowledge to the loop of scientific experiments. Thus, we instantiate our first computational Fact-Checker addressing the analysis phase of a Political Science experiment (Experiment 1). Figure 14 illustrates how this model is designed in the course of this chapter. We evaluate DeepDive's capacity to construct a local KB from heterogeneous data sources. The RDBMS holding the results of DeepDive's KBC process is represented in our model as the KB in the loop of Experiment 1. For this reason, the grounding process is based on SQL. Following a design choice further reviewed in the next chapter, we pack the Fact-Checker and the tracking of domain experts' rationales in a single workflow activity called "Results Evaluator", detailed in the course of this chapter.

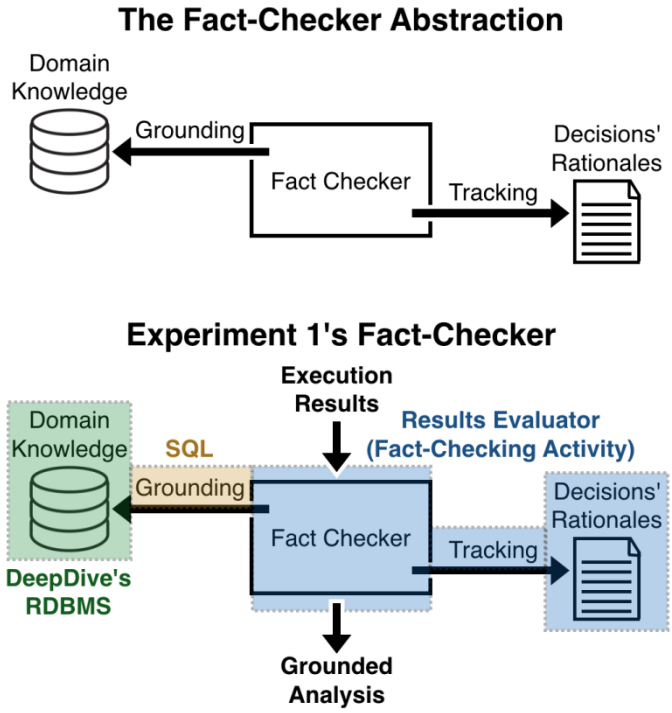

Figure 14 – Fact-Checker instantiated for Experiment 1, addressing the analysis phase.

With that said, this chapter starts with the preparation of DeepDive in order to run a KBC process for Experiment 1. Next, we cover Experiment 1's theoretical reference, before detailing its workflow activity-by-activity. Since our goal is to test our hypothesis in the analysis phase, we conclude by presenting and analyzing our approach's results in contrast with fact-checking performed by domain experts in the field of Political Sciences.

3.1 Preparing DeepDive

DeepDive may be further understood with the support of Experiment 1. To contextualize it, consider the Brazilian House of Representatives, where deputies can belong not only to parties but also to supra-partisan groups. Once after every election, DIAP (the Brazilian Inter-Union Department for Parliament Advisory) (DIAP - Departamento Intersindical de Assessoria Parlamentar 2016) releases several text reports analyzing the composition of supra-partisan groups. It aims to interpret the new House of Representatives regardless of partisan boundaries, and traditionally intends to fill the members of the following supra-partisan groups: the agribusiness group, the corporate group, the workers group, the women group, and the religious group. Therefore, the KB for this case consists in understanding six text reports by DIAP and populating facts of type "*deputy* d_i *belongs to supra-partisan group* g_j". Therefore, the schema of the target KB is specified by an ER graph `G = (E,R)` where `E` is one or more sets of entities (here, deputies and supra-partisan groups), and R is a set of relationships (here, a single relationship of type `belongs_to`).

In Experiment 1, DeepDive usage starts with a custom preprocessing step, which in this case is done over all DIAP reports. Similarly to what is done in DeepDive's official example (DeepDive - Tutorial: Extracting Mentions of Spouses from the News 2016), Python's Natural Language Toolkit (NLTk) (Bird 2006) is employed in order to perform basic natural language processing (NLP): first, to parse all sentences in the reports, as well as their respective words; and second, to tag each POS using NLTk's Mac-Morpho corpus (Aluísio *et al.* 2003), which is built upon an annotated set of news articles extracted from several Brazilian newspaper, contemplating over a million words. Additionally, for each one of those parsed sentences, it is important to identify

and mark up two sets of words that are going to be useful during entity extraction: the Brazilian states and the political parties acronyms. Therefore, for each preprocessed sentence, five types of attributes are generated: the raw sentence itself, detached from the body of text; the sentence's parsed words; these words followed by their tags from the POS-tagger; words identified as Brazilian states; and words identified as acronyms of political parties.

With the preprocessing step finished, it is necessary to implement two extractors: one for each entity in `E`, that is, deputies and supra-partisan groups. In DeepDive, extractors are basic units of domain-specific custom-made code. They are declaratively defined in DeepDive's configuration file, as illustrated in Figure 15. In the declaration of a given extractor, one should specify which extractors may be executed beforehand (i.e., its dependencies), the SQL query that may inject the input tuples, the table where results may be saved to, and any scripts providing UDFs that could be called during its execution. This design choice ensures that: most of the coding is expressed declaratively; extractors can perform fine-grained transformations by providing UDFs that operate in the level of a single tuple; extractors can be executed in a massively parallel fashion; users can maintain a clear chain of dependencies, and all data (i.e., both raw and transformed) gets transparently stored in a RDBMS.

That said, the first extractor is the deputies extractor, which applies some heuristics to extract mention candidates for entities of deputies. It assumes that deputies are often mentioned in reports next to their respective parties and states of origin. For broadening the mention candidates – they are further linked in the entity-linking process, making use of the provided ground truths –, the extractor considers as mention candidates all 1-gram, 2-grams and 3-grams before any matches given our heuristics (i.e., a party acronym followed by a Brazilian state or vice-versa). After implementing these rules in a Python UDF, the deputies extractor may be declared in DeepDive's configuration file, where information such as the query that feeds its input tuples and its output table are explicitly declared.

The second extractor is the supra-partisan groups extractor. Similarly, it applies heuristics to extract mention candidates for entities of supra-partisan groups. It assumes that such groups are characterized by the word "*bancada*" (Portuguese for "group")

followed by a word tagged by the POS-tagger as an adjective. Just as described for the deputies extractor, supra-partisan groups extractor is also declared in DeepDive's configuration file.

With all entity mention candidates extracted, it is possible to proceed to the extraction of candidates for the `belongs_to` relationship. For the purpose of this application, all co-occurrences of deputies and supra-partisan groups in a same section of a given report are considered candidates for this membership relationship. However, this extractor applies one additional filter: it removes candidates where the deputies are not found in the list of names selected from an external dataset (explained ahead as the votings dataset), at a Levenshtein distance (Navarro 2001) of two. Therefore, all facts candidates obtained from this process are implicitly grounded to that external dataset, which is the dataset for the workflow application. As in the previously described extractors, the relationship extractor is also declared in DeepDive's configuration file.

For the statistical learning task, a list of sixty true facts is provided to the DeepDive system, which is roughly 10% of all supra-partisan membership facts reported by DIAP. This list of true facts is the Machine Learning supervision for Experiment 1, though a substantially larger set of ground truths could be provided according to the rationale behind distant supervision. An additional extractor is implemented with the purpose of extracting generic features from a given sentence: it uses DeepDive's Python library called DDLib to issue generic features for each sentence stored by the relationship extractor. Using DDLib when dealing with text features is another inspiration from DeepDive's official example, and was considered a recommended approach towards learning linguistic patterns from text. Finally, to conclude the execution cycle, a simplistic inference rule is declared in DeepDive's configuration file, that forwards DeepDive prediction with no additional logic (i.e., the inference rule simply persists whatever DeepDive predicts).

Python user-defined function (UDF):

```
for row in sys.stdin:
    document_id, sentence_id, words_str, pos_tags_str = row.strip().split('\t')
    words = words_str.split(ARR_DELIM)
    pos_tags = pos_tags_str.split(ARR_DELIM)
    iterator = 0
    phrases = []

    while iterator < len(words) - 2:
        index = iterator
        bound = None

        if words[index].lower() == 'bancada':
           next_pos_tag = pos_tags[index + 1].split(',')[-1][:-1]
           if next_pos_tag == 'ADJ':
               bound = iterator + 2
               text = ' '.join(words[index:bound])
               length = bound - index
               phrases.append((index, length, text))
        iterator += 1

    for index, length, text in phrases:
        print '\t'.join(
            [ str(x) for x in [
                document_id,
                sentence_id,
                index,
                length,
                text,
                '%s_%s' %(sentence_id, index)
            ]])
```

Definition in deepdive.conf:

```
# Extractor 3: Extract mentions of supra-partisan groups
ext_groups {
  style: "tsv_extractor"

  # An input to the extractor is a row (tuple) of the following query:
  input: """
    SELECT document_id,
           sentence_id,
           array_to_string(words, '~^~'),
           array_to_string(pos_tags, '~^~')
    FROM   sentences
    """

  # Output of extractor will be written to this table:
  output_relation: "groups_mentions"

  # This user-defined function will be performed on each row (tuple) of input query:
  udf: "${APP_HOME}/udf/ext_groups.py"

  dependencies: ["ext_clear_table"]
}
```

Figure 15 – Detailed implementation of a DeepDive extractor (in this case, mention candidates for supra-partisan groups), including a declarative definition in deepdive.conf and its UDF in Python.

3.2 Theoretical Reference

Experiment 1 emulates the process by which a domain expert in the Political Sciences (i.e., a political scientist) would analyze the results of an unsupervised learning technique. In Machine Learning, unsupervised learning techniques (Witten *et al.* 2016) are based on the idea of performing a specific learning task relying solely on the

mathematical relationships among a given set of observations. In other words, in such techniques the ground truths are not accessible to the algorithms (i.e., the computer has not a single example of what would be a correct fact). In order to make sense of the results of unsupervised learning techniques, domain knowledge and fact-checking are of paramount importance. This aspect makes workflows that execute unsupervised learning techniques, regardless of the domain, potentially aligned to the motivation that introduces this research.

Liu *et al.* (2010) clarify how HIL is typically done for clustering tasks, which is an example of unsupervised learning. In clustering tasks, a data scientist may run several instances of one or more clustering algorithms, testing different configurations for their parameters. Internal clustering validation measures are typically used to assess the quality of clustering results from a strictly mathematical standpoint (e.g., how the internal densities of the clusters compare to their external densities). They oppose to external validation, which is basically domain expert's manual supervision. Therefore, data scientists may resort to three options in order to validate clustering results: they may rely on internal clustering validation measures; they may rely exclusively on extensive supervision from domain experts; or they may resort to a combination of both, narrowing the results space with an internal clustering validation measure before relying on a final and more selective round of supervision from domain experts. In this sense, the Silhouette index (Liu *et al.* 2010) is a very commonly used internal clustering validation measure.

In line with those theoretical references, we designed a scientific workflow for Experiment 1 whose goal is to arrange Brazilian deputies according to their voting behavior in a given set of legal bills. In this first case, the number of clusters (i.e., k) is a parameter that varies according to a parameter sweep (i.e., incrementally and within a predefined range of values). By testing different values for k, the goal is to find groups of deputies having most of their members from a single supra-partisan group. In turn, these supra-partisan groups are structured from DIAP reports and stored in the KB constructed in the last section. Experiment 1 leverages the high accessibility of this KB in the *post-mortem* of the workflow in order to evaluate clustering results for all values

of *k*, cutting what could be a very laborious external validation to be made by political scientists.

Experiment 1 is based on open public data from the Brazilian House of Representatives. The clustering task operates on top of data regarding how 384 deputies voted a set of 14 controversial bills. Since in Brazil there are plenty of parties (presently 35), supra-partisan groups are of great interest to domain experts. Thus, the clustering task is modeled after a KMeans algorithm (MacQueen 1967), as in the implementation of Scikit Learn library (version 0.17) (Pedregosa *et al.* 2011), where the *k* parameter varies within a predefined range starting from $k = 4$ to $k = 18$. This choice of values reflect the idea of finding political aggregations of the 35 parties, while knowing upfront that there are five traditional supra-partisan groups. Figure 16 illustrates the results of a KMeans execution with $k = 6$, projected to 2D using a mathematical transformation (PCA), so axes have no domain-specific meaning.

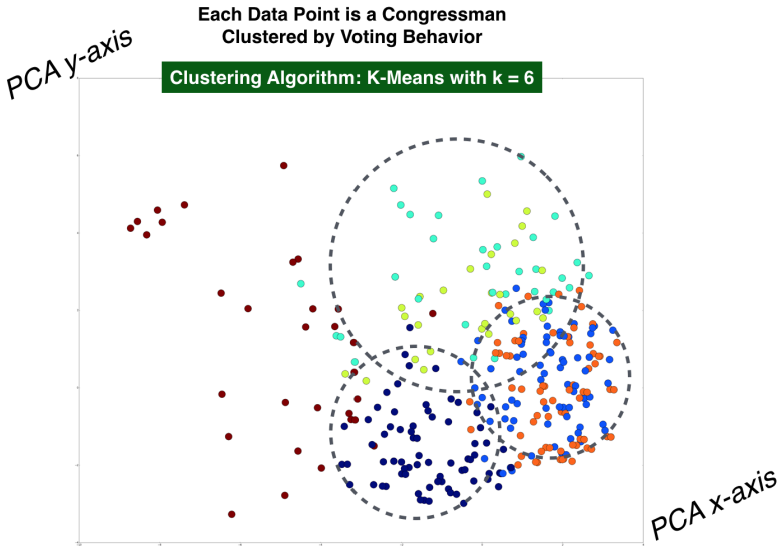

Figure 16 – Six clusters of deputies obtained with KMeans and projected to 2D using PCA.

3.3 Experiment Dataset

Data used in Experiment 1 is extracted through the API of the Brazilian House of Representatives (Dados Abertos - Legislativo 2016). The dataset covers 384 deputies voting 14 controversial bills – all from the same mandate –, regarding topics that include: a political reform, the reduction of the legal age for convicting adult criminals, and the relaxation of agribusiness regulations.

Considering the intention of running a clustering task (i.e., the premise of unsupervised learning is to explore mathematical relationships within a dataset), raw votes are converted to integer values. "YES" and "NO" votes are straightforward support or rejection towards a certain bill; "ABSENCE" could be the choice of deputies who are in fact attending a voting section but want to abstain from taking a position; "OBSTRUCTION" is limited to partisan leaders who want to boycott an ongoing voting section, so the meaning depends on the trend of a particular voting; and "NULL" values represent all the times deputies miss voting sections. Therefore, conversion of raw votes goes according to the following rationale: "YES" (+2), "OBSTRUCTION" (+1), "NO" (0), "ABSENCE" (-1) and "NULL" (-2). A basic characterization of votes distribution may be found in Figure 17, showing that "YES" and "NO" votes are significantly more important than the other options.

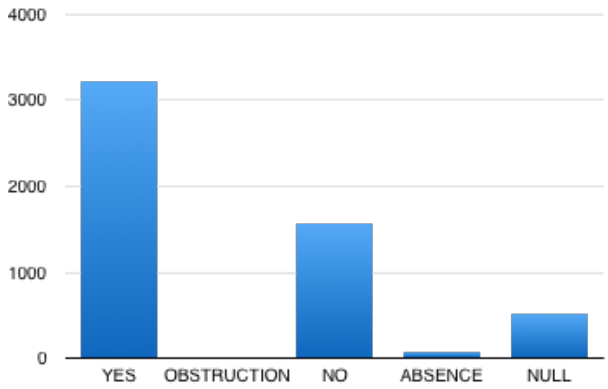

Figure 17 – Basic characterization of votes distribution.

3.4 Designed Workflow

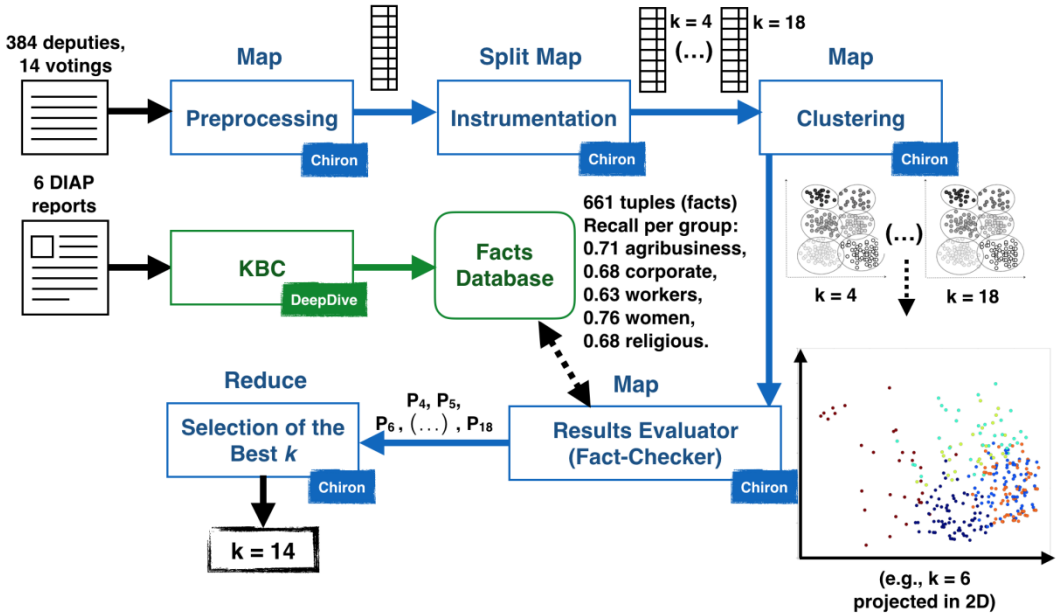

Figure 18 – Complete workflow of Experiment 1 (Bursztyn, Dias and Mattoso 2016).

The complete workflow of Experiment 1 is described in Figure 18. It comprises the following steps:

1. Six unstructured reports from DIAP are considered sources of ground truth, so they are futurely handled as input to DeepDive;

2. Before providing it to DeepDive, a preprocessing step ensures texts are parsed and tagged in ways they become suitable to the developed DeepDive extractors;

3. Then, extractors obtain mentions and entities for deputies and supra-partisan groups;

4. Upon that, DeepDive grounds deputies to the names recorded in the votings dataset and simply populates a relation of type `belongs_to`, covering roughly 70% of the desired supra-partisan groups;

5. The non-KBC part of the workflow (marked with the "Chiron" label) starts by executing two activities: one for reading the votings dataset and mapping votes to

integer values, and the other for preparing 15 executions with different configurations for the *k* parameter (i.e., varying *k* from 4 to 18);

6. Then, a clustering activity performs the KMeans algorithm with the provided configurations;

7. For each value of *k*, the Results Evaluator activity evaluates clustering results according to the KB populated by DeepDive, culminating in a quality measure for that *k*. This quality measure is the average precision considering all clusters. For each cluster, we calculate its homogeneity: we find the most numerous supra-partisan group represented in it and calculate the proportion of deputies in that cluster who indeed belong to this most prominent supra-partisan group (i.e., this is the fact-checking activity);

8. Finally, the workflow finishes by indicating the value of *k* that is associated to the maximum quality measure, meaning that it generated the purest clusters: on average, clusters that are more dominated by members of a single supra-partisan group.

3.5 Results and Conclusions

As clarified by Figure 19, the quality measure varies until it reaches a maximum average precision for $k = 14$, conforming to the facts database (again, the fact-checking activity is linked to results from DeepDive). If domain experts had to navigate through clustering results for all *k* values tested, they would have to assess the membership of 384 deputies with respect to five supra-partisan groups 15 times.

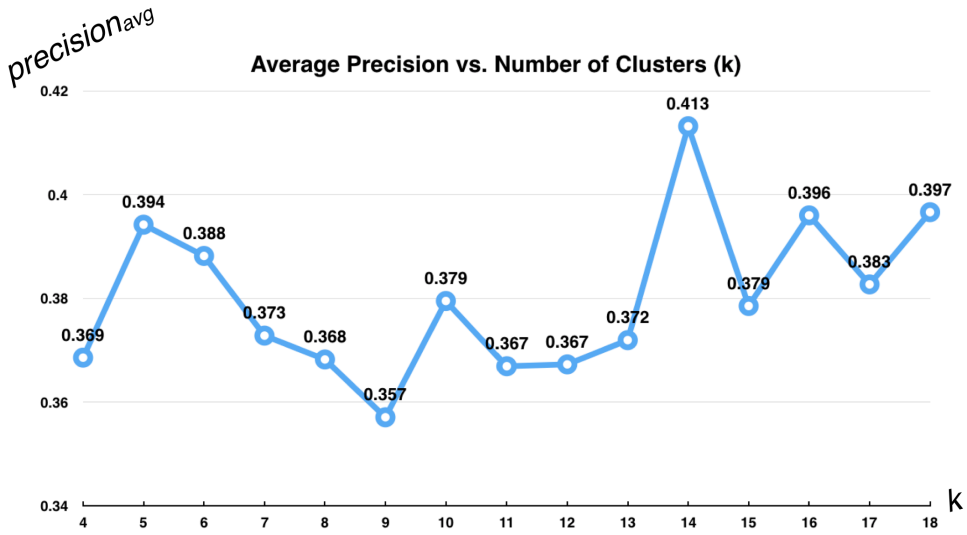

Figure 19 – Clustering precision varying with k in range [4,18], according to the facts database.

With 384 x 15 = 5760, it would mean almost *six thousand assessments* involving the `belongs_to` relationship, which probably would require domain experts to consult the original DIAP reports holding a total of 582 membership facts (i.e., deputies belonging to supra-partisan groups). Therefore, at a first glance, it is clear that encapsulating domain-specific knowledge using KBC and bringing it to the loop of experiments that perform unsupervised learning tasks are productive ideas. As seen in this set of results, benefits could range from saving a substantial amount of time with external validation to transforming a virtually unfeasible validation into a practical and reliable computational activity. Moreover, Experiment 1 could be re-executed with a different voting dataset (e.g., approaching an entirely different set of legal bills), and would not require more human hours in order to find the new best *k*.

Complementarily, it is important to compare the proposed approach to a validation strategy that intelligently combines the use of an internal clustering validation measure followed by an external validation procedure. After all, it is possible to narrow the results space before requiring participation of domain experts in the loop of the experiment. Similarly to the DeepDive-based evaluation, Figure 20 shows the average Silhouette index for each one of the values tested for *k*. As other internal clustering validation measures, the Silhouette index is a strictly mathematical measure that

compares the average distance for the data points inside a given cluster to the average distance of those outside (Liu *et al.* 2010). Thus, an average Silhouette index implies having this cluster-specific measure applied to all clusters generated by a certain *k* and calculating the average value.

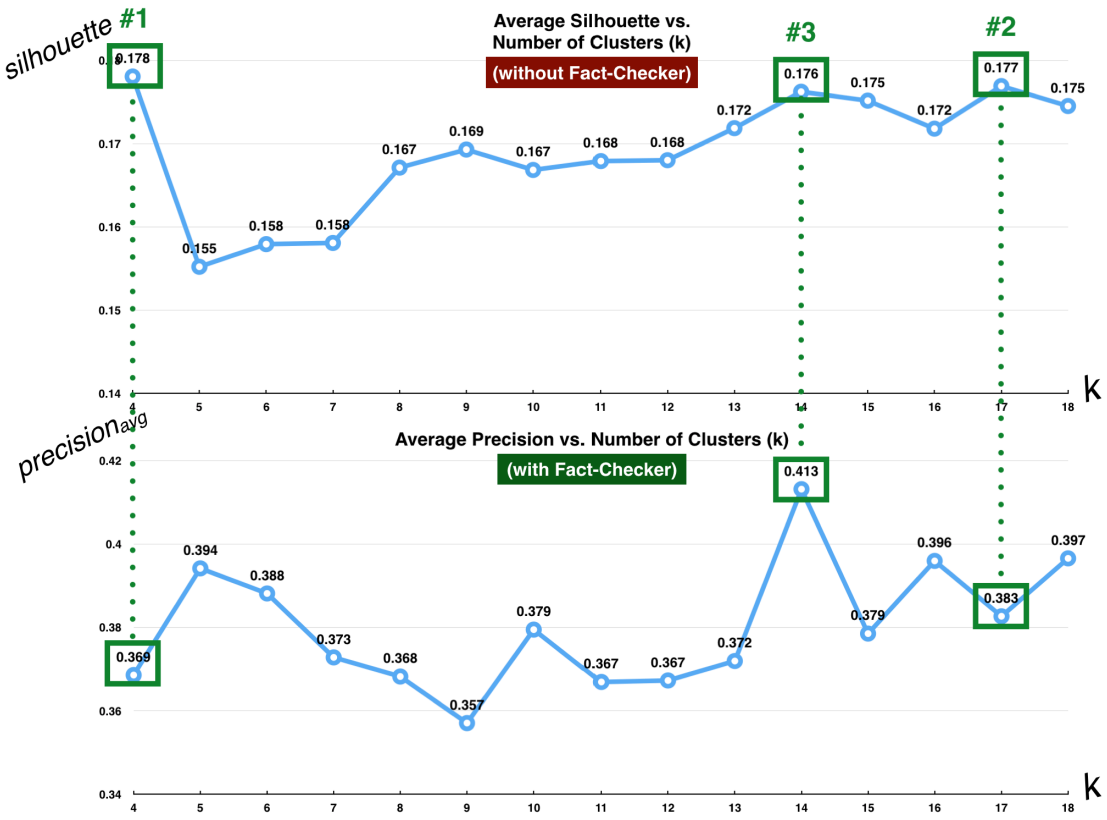

Figure 20 – Comparison between an internal clustering validation measure and complete external validation as enabled by KBC.

If domain experts were to pick the clustering results associated to the top three average Silhouette for a thorough human validation, they would end up with clustering results for $k = 4$, $k = 17$, and $k = 14$, in this order. It is worth-noticing that the best value for *k*, which is $k = 14$, would barely make the cut. This has two implications: first, that domain experts would still have to make more than *one thousand* assessments, which is still very laborious and error-prone; and second, as shown by the fact that the best value of *k* was almost left off, the success of this combined validation strategy strongly depends on the cut to be made. After all, there is no guarantee of correlation between a

strictly mathematical measure, such as the Silhouette index, and the actual ground truths. It is expected to be a simple heuristic for narrowing the results space, based on a clear trade-off between the amount of human hours available (or willing to be invested) and the likelihood of success. For this particular case the heuristic performed poorly, which strengthens the case for domain knowledge made highly available by means of KBC. More generally, it strengthens the case for KB having a role in the HIL process.

Finally, it is worthy of note that KBs could be an effective basis for narrowing the processing space, which consequently saves computational resources. The following strategy is also known as a parameter sweep with varying resolution. Looking again at Figure 19, it is possible to make a two-fold division in the set of values for *k* around its median value (i.e., $k = 11$). This procedure generates two subsets with seven values each: the ones before the median (i.e., $k = 4$, $k = 5$, …, $k = 10$), and those after the median (i.e., $k = 12$, $k = 13$, …, $k = 18$). It is possible to narrow the processing space by taking a rough 50% sample in both subsets, which could be done by selecting either the even or the odd values in both subsets. Either way, the average quality in the second subset would be superior to the first subset's: if even values are considered (four values on both sides), $0.393 > 0.376$; instead, if odd values are considered (three values on both sides), $0.378 > 0.374$. In a parameter sweep with varying resolution, the workflow would use the second iteration to increase the resolution and focus on the unsampled values from the second subset. The workflow would eventually converge to $k = 14$, making the strategy consistent with the full-fledged execution. With this configuration, the workflow would discard roughly 25% of the processing space: in this case, it would discard three or four clustering steps with no loss on the convergence. As the range of the parameter sweep widens, the strategy enabled by a KB would become arguably more appealing. Therefore, in addition to saving domain experts time, Experiment 1 shows that bringing domain-specific knowledge to the loop has the potential to directly save HPC resources.

It is true that current state-of-the-art in KBC adds an overhead of work and complexity to the stack of scientific technologies. However, as KBC systems have become more user-friendly and mature, they have reached a stage where a single researcher can fully operate them and deploy a KB, thus making that overhead very

manageable. Although in this particular setting it could be possible to attain the same KB in a manual process (i.e., reading each report by DIAP, extracting deputies name by name with their respective supra-partisan groups, and storing these facts in a structured fashion), several domains are reaching a point where this is too laborious or no longer feasible. This is what justifies the increasing number of online domain-specific KBs, which are live repositories of domain facts, and this is why we pose Experiment 2 in the Pharmacogenetics domain – covered in chapter 5.

Based on the results of Experiment 1 it is possible to conclude that:

● The construction of a KB in the Political Sciences *saved nearly six thousand assessments* that would require domain experts to consult the original unstructured reports storing a total of 582 membership facts (i.e., deputies belonging to supra-partisan groups). Integrating easy-to-use KBC systems in experiments that perform unsupervised learning tasks and rely on domain expert validation can save a substantial amount of time with external validation, and transform a virtually unfeasible validation into a practical and reliable computational activity.

● We compared our KBC strategy to an alternative strategy that applies a heuristic for narrowing the results space. We analyzed possible cuts in the results space by selecting the most promising values according to the Silhouette Index, which is a popular internal clustering validation measure. We found that the heuristic performs poorly for Experiment 1. It not only maintains the dependency on human labor but does it at risk of cutting out promising parts of the results space. This conclusion matches the theoretical reference because there is no guarantee that an internal clustering validation measure, which simply captures mathematical relationships in the data, necessarily encodes domain-related judgement.

● With a KB in place, Experiment 1 can be re-executed with an entirely different voting dataset (e.g., approaching an entirely different set of legal bills), and that would not require more human hours in order to find the new best *k*.

Chapter 4 – Annotation of Data and Processes in Interactive Computing

In chapter 2 we reviewed related works that were fundamental to the design of experiments that tested our hypotheses in the analysis phase as well as in the composition phase (chapters 3 and 5, respectively). We also delineated research opportunities upfront. As anticipated, Experiment 2 motivates the development of a complementary library to improve the tracking of users' sessions in Jupyter Notebooks. In this chapter we present the features and explain corresponding design choices in the development of jupyterADP library (or jADP)[1], the Python library for Annotation of Data and Processes in the context of Notebooks. We also show real examples of jADP's "trials files", which are particularly easy to handle once they are based on Python's native data structures. These files preserve both high-level and low-level resources regarding scientific data and scientists' decision processes while involving KBs in the loop of an experiment. From the interactions made by scientists to the consolidated in-memory dataframes, everything can be preserved. Finally, closing this chapter, we revisit Experiment 1 and review the Fact-Checker model as it was initially designed (Bursztyn, Dias and Mattoso 2016).

4.1 jADP: Jupyter Annotation of Data and Processes

While implementing a Jupyter Notebook with the purpose of integrating and manipulating domain knowledge scattered across online KBs, in the context of HIL of scientific experiments, seven key operations related to versioning, annotation and basic provenance of data were identified. They are briefly described as follows:

[1] https://github.com/vbursztyn/jupyterADP

● `load_dataset` – Annotating the load of a static dataset: this represents all static data that is accessible for bulk download from online KBs, or any other auxiliary domain-specific data. Such datasets could be included in the Docker container storing a Jupyter Notebook, as hypothesized previously in this chapter. A global configuration flag can force jADP to automatically save all the raw files associated to this type of annotation.

● `register_url` – Annotating an external API: the goal of this operation is to record a brief profile for all APIs that may be called throughout the Notebook. If there are multiple KBs to be accessed, or multiple namespaces within a given KB, it could be semantically relevant to have a brief profile defining the scope of that API.

● `query_url` – Keeping track of API calls: complementing the previous operation, this operation allows saving all external calls made by a unit of code in the Notebook. It necessarily refers to one single API annotated using the previous operation and to a relative path (typically, a resource in RESTful APIs); optionally, it can record all parameters used in such calls.

● `modify_data` – Keeping track of relevant ETL: the goal of this operation is to save relevant transformations made over intermediate data. Considering Python in-memory dataframes (a type of data table regularly used in ETL operations, both in Python and in R), this operation allows to save column names and unique ids of rows. Therefore, joined, projected and selected dataframes can be saved in a more abstract level.

● `save` – Keeping track of relevant dataframes: the goal of this operation is to save the entire contents of relevant intermediate data. It complements the previous operation as it allows saving all cells in a dataframe.

● `interaction_by_scientist` – Keeping track of all interactions made by scientists: the goal of this operation is to save higher level decisions made by domain experts using interactive components in the Notebook. For the case of the search field shown in Figures 12 and 22, this operation would allow to save the keywords used in the searches along with a column it is associated to (i.e., in the case of Figures 12 and 22, the column storing information on diseases – the summaries).

● `run_workflow` – Annotating calls to external workflows: once a data-driven decision is made, a possible step would be to dispatch a data-intensive workflow in a HPC environment with restrict access to the external world. The goal of this operation is to accurately save whenever external workflows are called, recording details about the workflow itself, the chosen parameters, and the HPC environment running the job.

A final `commit` operation must be called in order to dump all of jADP in-memory records to a ADP (annotation of data and processes) file, which is Python pickle file (i.e., a binary file) logging the seven types of operations in a JSON-like format indexed by timestamps. All ADP files are automatically placed in a subdirectory named "trials", which numerals representing the sequence of executions.

Figure 21 shows how jADP is imported and instantiated in a Jupyter Notebook in the context of Experiment 2. It also shows real usage for the `load_dataset`, `register_url`, and `modify_data` operations.

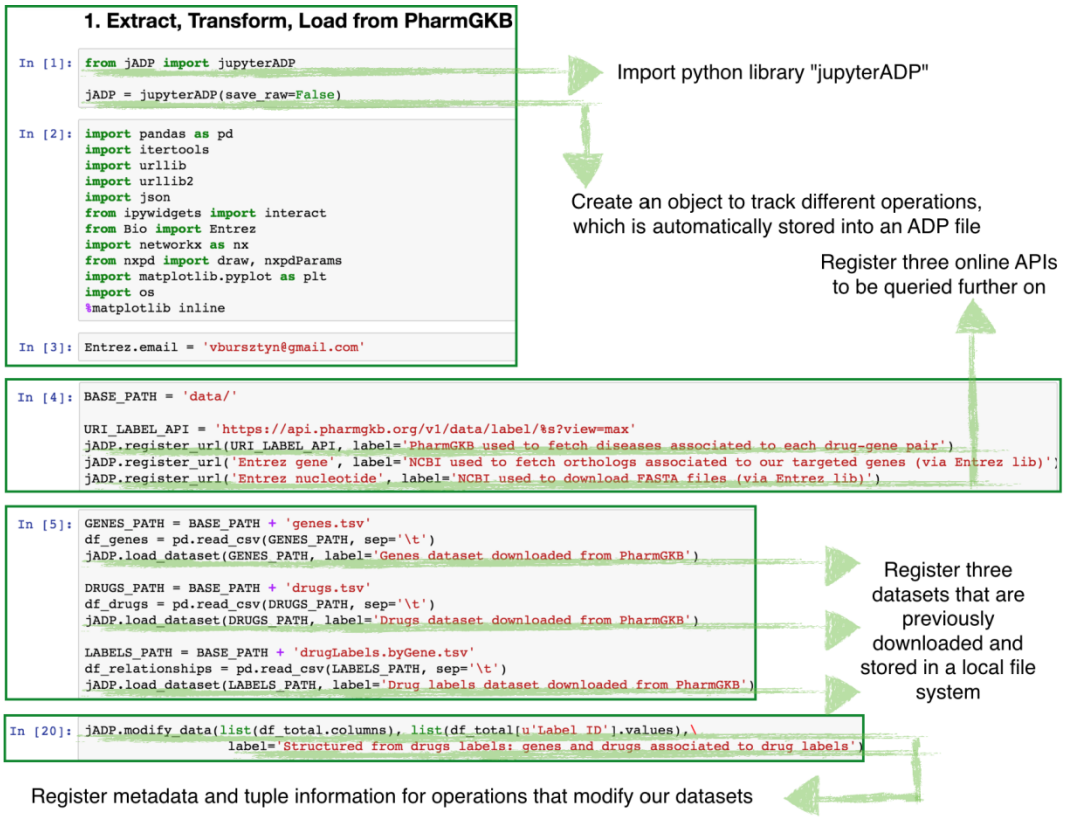

Figure 21 – jADP instantiation in Experiment 2.

Figure 22, again in the context of Experiment 2, shows real usage for all the other operations: `query_url`, `save`, `interaction_by_scientist`, `run_workflow`, and `commit`, which generates an ADP file in the "trials" subdirectory.

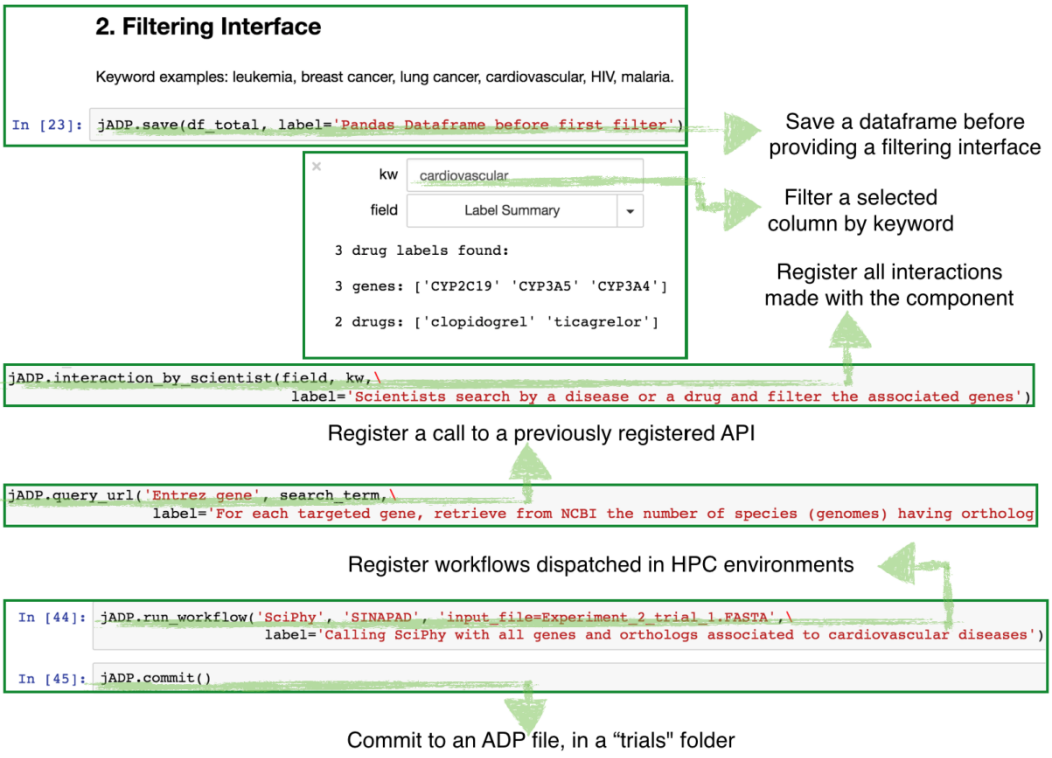

Figure 22 – jADP operations in Experiment 2.

4.2 Features of an ADP File

Figures 23 and 24 illustrate the anatomy of a real ADP file. Figure 23 shows that opening a trial is straightforward, costing only a couple of lines of Python code. Next, it shows that all the seven types of operations are easily accessible: `DATASETS` holds details on `load_dataset`, `URLS` to `register_url`, `QUERIES` to `query_url`, `ETLS` to `modify_data`, `RESULTS` to `save`, `INTERACTIONS` to `interaction_by_scientist`, and `RUNS` to `run_workflow`. In this case, three loaded datasets are retrieved, 18 ETL operations, and 20 interactions made by scientists – all indexed by timestamps. By opening a specific interaction, it becomes clear that a

scientist filtered diseases with the keyword "cardiovascular" at that given time. Further on, in Figure 24, it is possible to visualize how jADP is able to store hard copies of important dataframes whenever a `save` operation indicates so. Finally, it is also possible to retrieve a call to an external workflow (the SciPhy workflow – Ocaña *et al.* 2011), which is the ultimate goal of Experiment 2.

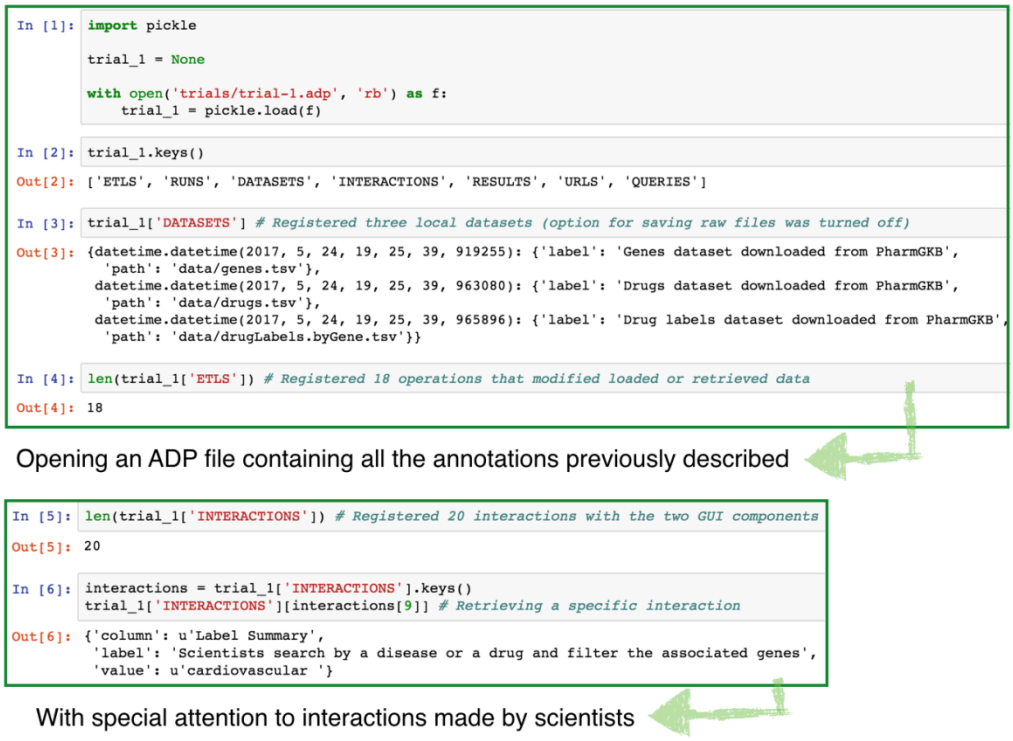

Figure 23 – Opening an ADP file and retrieving an interaction.

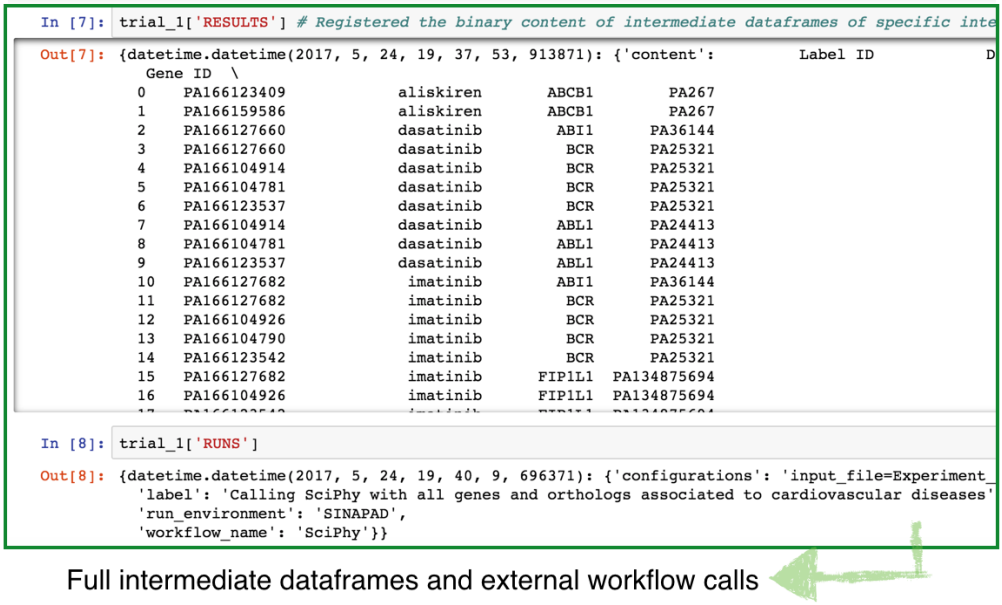

Figure 24 – Retrieving a hard copy of a dataframe and a parameterized workflow execution.

4.3 jADP in Experiment 1

Revisiting the Fact-Checker model in chapter 3, it is possible to visualize how jADP could help to better organize Experiment 1's fact-checking process. Figure 25 illustrates the improved design. It encapsulates the original Results Evaluator activity in a Jupyter Notebook that is annotated using the jADP library. This allows for the preservation not only of the data used during fact-checking (e.g., the exact tuples returned by the queries to DeepDive's RDBMS), but also of all decisions that were previously hard-coded in Experiment 1's Results Evaluator activity. For instance, in Experiment 1 the quality of clustering results was evaluated according to the homogeneity of clusters (i.e., their precision w.r.t. one of the supra-partisan groups). This is a rationale that should be stored in an easily recoverable way in case other criteria were to be tested in the future of the experiment. Therefore, in an improved design, Experiment 1's workflow would end before the Results Evaluator activity, whose role would be assigned to a Jupyter Notebook supported by the jADP library.

This Notebook would be responsible for finding the best k while preserving all relevant parameters in that particular execution of the fact-checking process (e.g., that k was chosen based on a specific definition of precision). Besides being more efficient in the fact-checking process, as seen in Experiment 1, this model would provide a tracking capacity that is likely to be overlooked in human fact-checking.

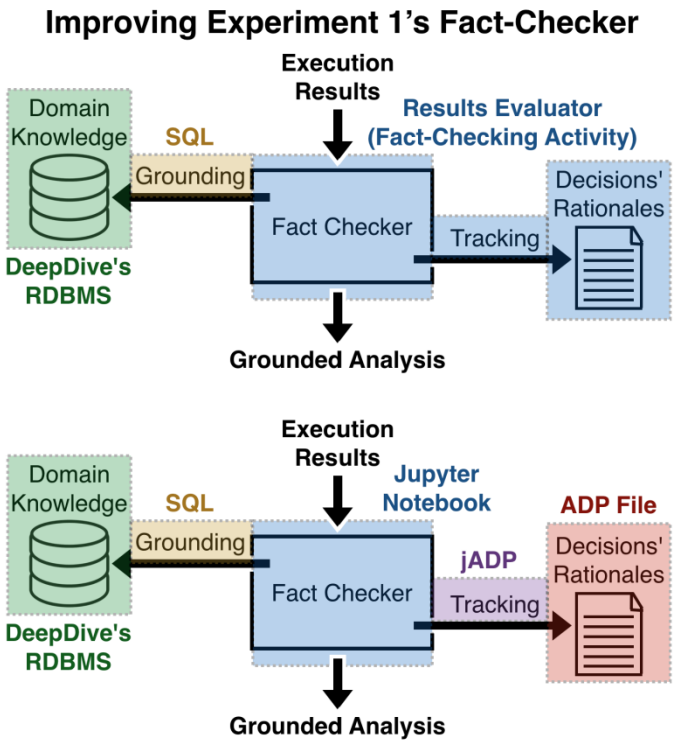

Figure 25 – Using jADP to improve Experiment 1's Fact-Checker model.

Chapter 5 – Online KBs to Support Fact-Checking in the Composition Phase

After testing our hypothesis in the analysis phase, we instantiate our second computational Fact-Checker, addressing the composition phase of a Pharmacogenetics experiment (Experiment 2). Figure 26 illustrates how this model is designed in the course of this chapter. In Experiment 2, we test the scenario where domain knowledge is maintained in a set of remote KBs made available online, which is a common trait in Bioinformatics but also represents a trend in several other domains (DeepDive - DeepDive Applications 2016). These KBs are represented in Figure 26, for simplicity, as a single KB in the loop of Experiment 2. Consequently, the grounding process is generically represented as a process done through APIs. Differently from Experiment 1, Experiment 2 separates the Fact-Checker execution code and the parameters describing the rationales behind domain experts' decisions. This tracking process is based on a custom-made library (jADP) placed in the context of Jupyter Notebooks, as addressed in the previous chapter.

With that said, this chapter starts by covering Experiment 2's theoretical reference, before diving into its workflow in details. Since our goal is to test our hypothesis in a composition phase with strong dependency on fact-checking, we conclude by presenting and analyzing our approach's results in contrast with fact-checking performed by domain experts in the field of Pharmacogenetics.

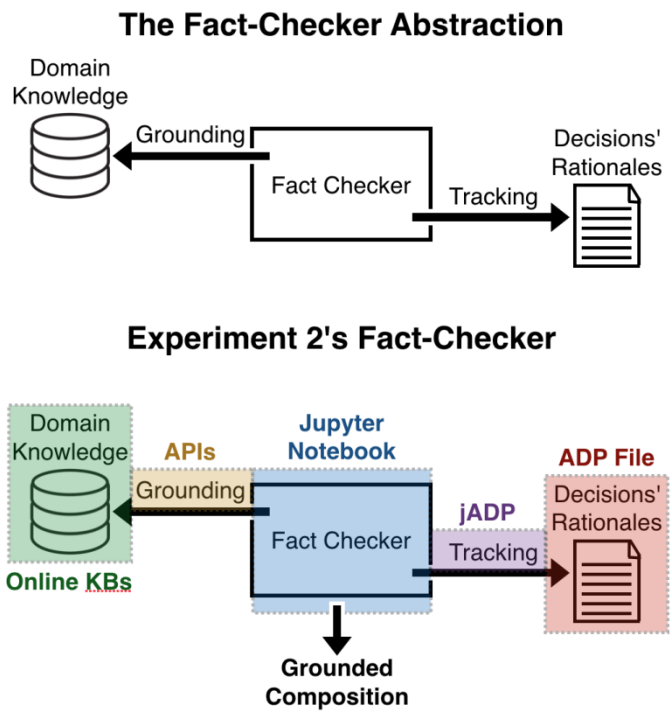

Figure 26 – Fact-Checker instantiated for Experiment 2, addressing the composition phase.

5.1 Theoretical Reference

Pharmacogenetics is the area that studies the relationships between drugs and genes. Similarly, Pharmacogenomics studies the relationships between drugs and genomes. Frequently, phenotypes of diseases or medical conditions are included in Pharmacogenetic analyses, typically as a criterion to select a set of genes on which further studies may be conducted. Therefore, Pharmacogenetics is an area where relationships between drugs, genes and diseases have central role.

In parallel, Phylogenetic analysis is another field encompassed by Bioinformatics, which aims at comparing hundreds of different genomes in order to identify the evolutionary similarity between different organisms. Phylogenetic could be performed over a set of genes originated from a particular study in Pharmacogenetics. In this context, targeting a particular disease, domain experts can investigate new approaches for the development of treatments considering aspects of evolution, genes similarity and genomes similarity.

Diverse types of Bioinformatics-related applications are utilized in Phylogenetics, such as Multiple Sequences Alignment (MSA) and Evolutionary Model Election, which are growing in scale and complexity. Managing phylogenetic experiments is far from trivial, once the associated activities are computationally intensive and generate large amounts of data. This class of experiments is likely to benefit from scientific workflows. Particularly in workflows for phylogenetic analysis, scientists perform a well-defined sequence of activities so to produce a set of phylogenetic trees, which in turn allow for the inference of evolutionary relationships between genes from divergent species.

SciPhy (Ocaña *et al.* 2011) is an example of workflow for phylogenetic analysis. SciPhy performs multiple parameter sweeps and executes all activities for each input file in a given set of input files. These activities are the four listed as follows: (i) multiple sequences alignment (MSA), (ii) conversion of alignment, (iii) evolutionary model election, and (iv) the construction of phylogenetic trees. They execute the following sets of Bioinformatics-related applications and tools, respectively: programs related to MSA (MAFFT, Kalign, ClustalW, Muscle and ProbCons), Readseq, ModelGenerator, and RAxML. SciPhy is briefly depicted by Figure 27.

A typical parallel execution of SciPhy may last more than a couple of days in a HPC environment. The execution in such environments (e.g., scientific clouds) typically focuses on optimizing the scheduling of activities or on finding the best data distribution policies, which are fundamentally important. More recently, dynamic steering using SWfMS also comes at play (Dias 2013).

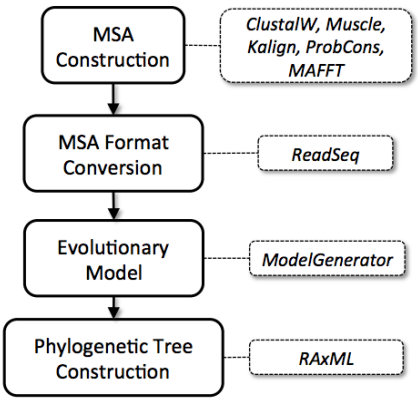

Figure 27 – The SciPhy workflow (Dias 2013).

To illustrate, for the MSA activity, there are diverse alternative implementations, such as MAFFT, ProbCons, Muscle, ClustaW and Kalign). These programs may be considered interchangeable, but the quality of their results depends on certain characteristics of their input data. Each alternative could have a different impact on execution in terms of performance or quality of results. Some workflows include all alternatives for MSA because the best choice may not be possible beforehand. However, during execution, based on the analysis of partial results, the scientist may realize that the current alternative is unsuitable for the corresponding input data. There is no purpose on continuing execution "as is" once the scientist is able to steer it towards a potentially more suitable alternative, instead of waiting for all the range of alternatives. For example, consider that the execution of SciPhy starts by utilizing MAFFT for the MSA activity. After 12 hours of runtime the scientist receives a notification from the execution machine informing that execution may take more time than expected. This could be an insight for the scientist to act on the execution in order to satisfy a time constraint. In a nutshell, the scientist should be able to change the program associated to the MSA activity (for instance, utilize ProbCons instead of MAFFT) so to obtain an alignment in a more cost-effective fashion.

5.2 Experiment Dataset

The PharmGKB (Pharmacogenetics Knowledge Base) (Hewett *et al.* 2002) exists since 2001 and have gathered genomic, phenotype, and clinical information collected from pharmacogenetic studies performed by the various authorities in the domain. It includes tools for browsing, querying, editing and downloading the information – a wider set is available for registered members while a limited subset is made available publicly. It started with studies comprising 150 human genes and now it comprises at least 619 human genes, as found in our research. The richness of PharmGKB lies mostly in the relationships that exist between genes, drugs and medical conditions, which are typically documented in a variety of data sources.

In addition to PharmGKB, the National Center for Biotechnology Information (NCBI) has also invested in a number of projects motivated by facilitating researchers'

access to domain knowledge. NCBI's project "Entrez" (Maglott 2005) provides an unified search interface for many discrete health sciences databases, from a gene-centered angle. Through this interface, domain experts can search for genes and access information derived from various types of research, from computational inference of similarity to experimental research. For the purpose of Experiment 2 and in the context of the SciPhy workflow, two sorts of gene-centered information are valuable: first, in what other species similar genes can be found (i.e., what species carry ortholog genes); and second, the gene's sequencing information (i.e., its genetic content), commonly referred as its FASTA file.

Related to the data manipulations made in the context of Experiment 2, previous works have created queryable interfaces by building on top of PharmGKB and NCBI (Yu *et al.* 2009) or created network visualizations from other knowledge bases in Bioinformatics (Glaab *et al.* 2012).

5.3 Designed Workflow

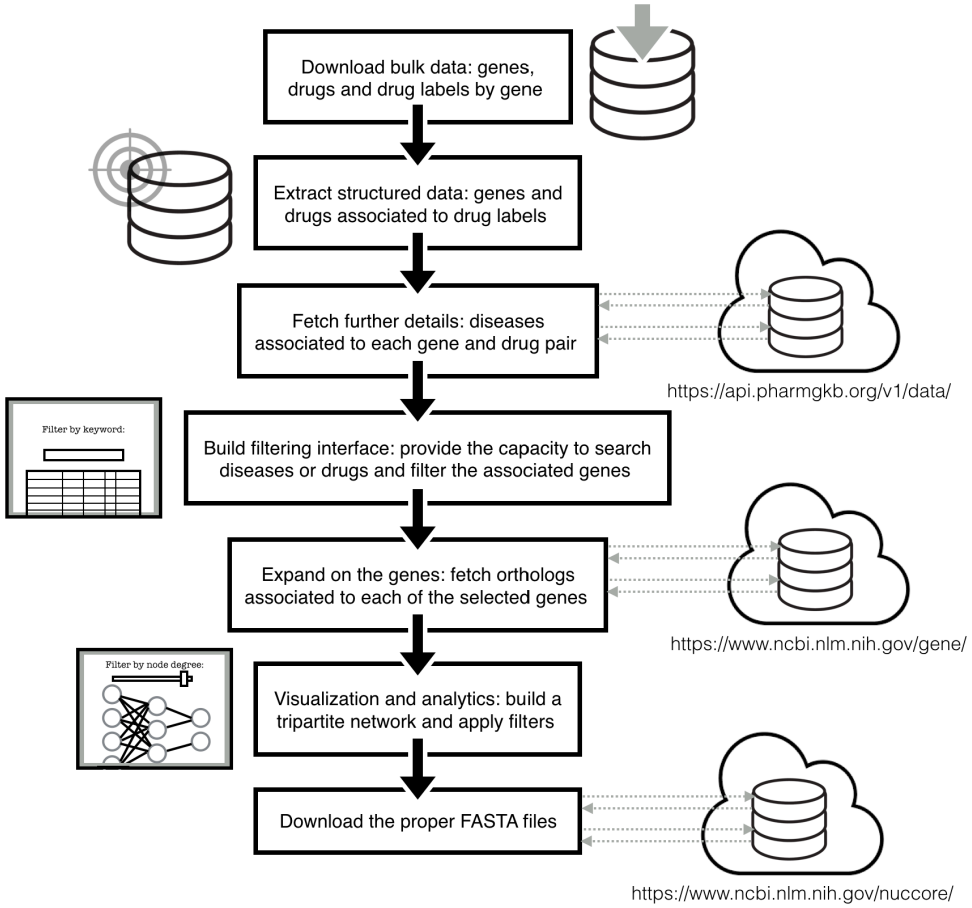

Figure 28 – Complete workflow of Experiment 2.

Experiment 2's complete workflow is described in Figure 28. Its activities may be depicted in details as follows:

1. Three static TSV files (tab separated values) are publicly available in PharmGKB for bulk download. They comprise the set of detailed genes, the set of detailed drugs, and the set of drug labels, which represent the tests conducted by authorities in the Pharmacogenetics domain, linking a given drug to a set of associated genes. Drug labels are semi-structured: although these relationships are described in plain text, such text spans follow well-defined rules;

2. Therefore, the next activity aims at extracting structured information from the raw semi-structured dataset: it performs ETL so to produce clear links between drugs and associated genes;

3. With clear relationships between drug label IDs, drug IDs and gene IDs, it is possible to fetch additional data that is not made available for bulk download, but does exist in PharmGKB's RESTful API. By querying PharmGKB's RESTful API, specifically the resource for drug labels (i.e., by iterating over all drug label IDs), it is possible to retrieve the summary of each drug test. This summary details in plain text how the drug associated to those genes may be recommended for a particular disease or medical condition;

4. It is worth noticing that those summaries are one of the benefits delivered by PharmGKB, as it avoids the cost of reading drug testing reports scattered across data sources, just to gather the facts of interest (verified relationships between drugs, genes and medical conditions). As stated, drug testing can be performed by several authorities, so having the facts summarized and in a centralized repository represent two-fold benefits. Summaries serve as basis for the first interaction point with domain experts: using an interactive GUI component in a Jupyter Notebook, scientists can search for a keyword (typically, a disease or a medical condition) and filter all genes and drugs related to it;

5. By applying that first filter, scientists are explicitly defining the scope of their experiments, thus making the searched keywords semantic markers of great value. By

moving to the next activity, Experiment 2 exits the context of PharmGKB and enters the context of NCBI. This activity fetches NCBI's repository of genes through the Entrez's Python library and collects all orthologs found for the set of filtered genes (genes associated to a meaningful research subject – typically a disease –, looked up in the summaries). Having the orthologs organized in a single KB is another benefit worthy of note, this time delivered by NCBI's projects. A set of human genes filtered after a disease can have up to thousands of ortholog genes spread across hundreds of other species (mapped genomes). Therefore, this activity shows the number of orthologs and leaves the decision to fetch their details up to the scientists;

6. If their details are fetched, a tripartite network visualization (Figure 29) shows how species (pink nodes) carry ortholog genes for the filtered genes (green nodes), which in turn are associated to drugs (blue nodes) found for that particular disease, as defined in the first interaction. This visualization allows for a second interaction, also shown in Figure 29. In the same Jupyter Notebook, scientists can use a second interactive GUI component to render new versions of the network visualization. They can study which species carry more or less ortholog genes given the set of filtered genes (Figure 29 shows the genes filtered for the keyword "cardiovascular"). Scientists can see that four species have at least two ortholog genes given that set of three genes. After interacting with the threshold bar, scientists now see that only two species have ortholog genes for all genes in the study. Scientists' interactions at this point also provide a valuable semantic marker for the experiment being composed.

After selecting genes associated to cardiovascular diseases, scientists can interact with and analyze the relationships between species (genomes), the associated genes and associated drugs:

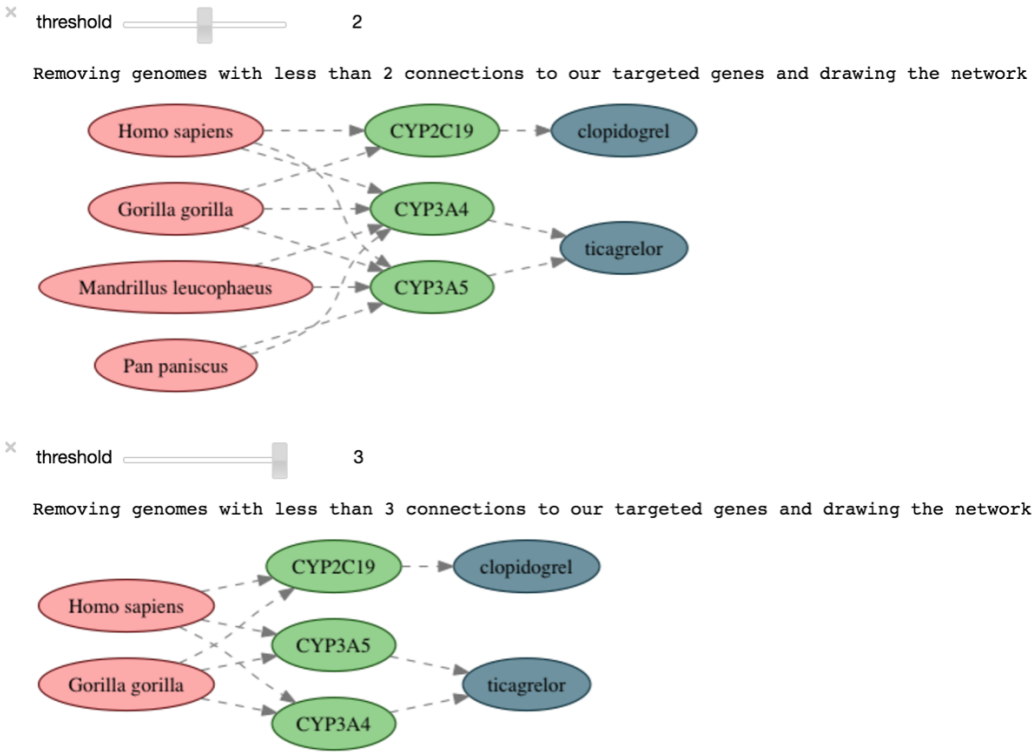

Figure 29 – The second interaction extends scientists' analytical capacity, as they can visualize filtered genes, species carrying ortholog genes and all associated drugs. An interactive bar allows cutting out species according to scientists' criteria.

7. Finally, with all the analytical capacity enabled by PharmGKB, NCBI and the Jupyter Notebook, scientists can perform one last activity and download the associated FASTA files from NCBI. This activity may also send the concatenated input file to a HPC environment, such as Brazil's SINAPAD (Gadelha 2006).

5.4 Results and Conclusions

This workflow was tailored in partnership with Kary Ocaña, a domain expert in the field of Bioinformatics. It was designed to automate some of the most laborious tasks faced by Kary in her research process. In (Ocaña and Dávila 2011), a manual spreadsheet was created and maintained in order to support the composition phase of her experiment. Figure 30 shows how this manual compilation, which strongly relied on

NCBI's web search interface, reaches remarkable complexity in practice. In Figure 30, columns represent genes (referred unintelligibly by their internal IDs) and rows represent species both by their intelligible names and the corresponding IDs. With the support of such spreadsheet, species are studied in terms of their potential on carrying ortholog genes, being the basis for Kary's decisions towards including or excluding those specific genomes in her experiment.

When the composition of an experiment is strongly based on fact-checking through existing KBs, it is clear that using an interactive computing environment to tap into those KBs and bringing that knowledge to the loop are productive ideas. As seen in Experiment 2, benefits could range from saving a substantial amount of time with the complexity of the composition phase to preserving the high-level rationales behind domain-specific decisions. Consequently, given that the underlying KB is a live repository of domain facts, Experiment 2 could be re-executed at any point in the future. In the case of Figure 30, in one year from now, Pharmacogenetics authorities could have found additional genes associated to the keyword "cardiovascular." Furthermore, species whose genomes are poorly covered in the present could have a better coverage in this future scenario, changing the results from the rationale that included or excluded ortholog genes coming from certain species. This could prompt a substantially different execution which could be done automatically (i.e., with a programmatic trigger), yet maintaining the original subject of study. Figure 31 shows how jADP’s operations, depicted in the previous chapter, enable the tracking of all domain-specific decisions in Experiment 2.

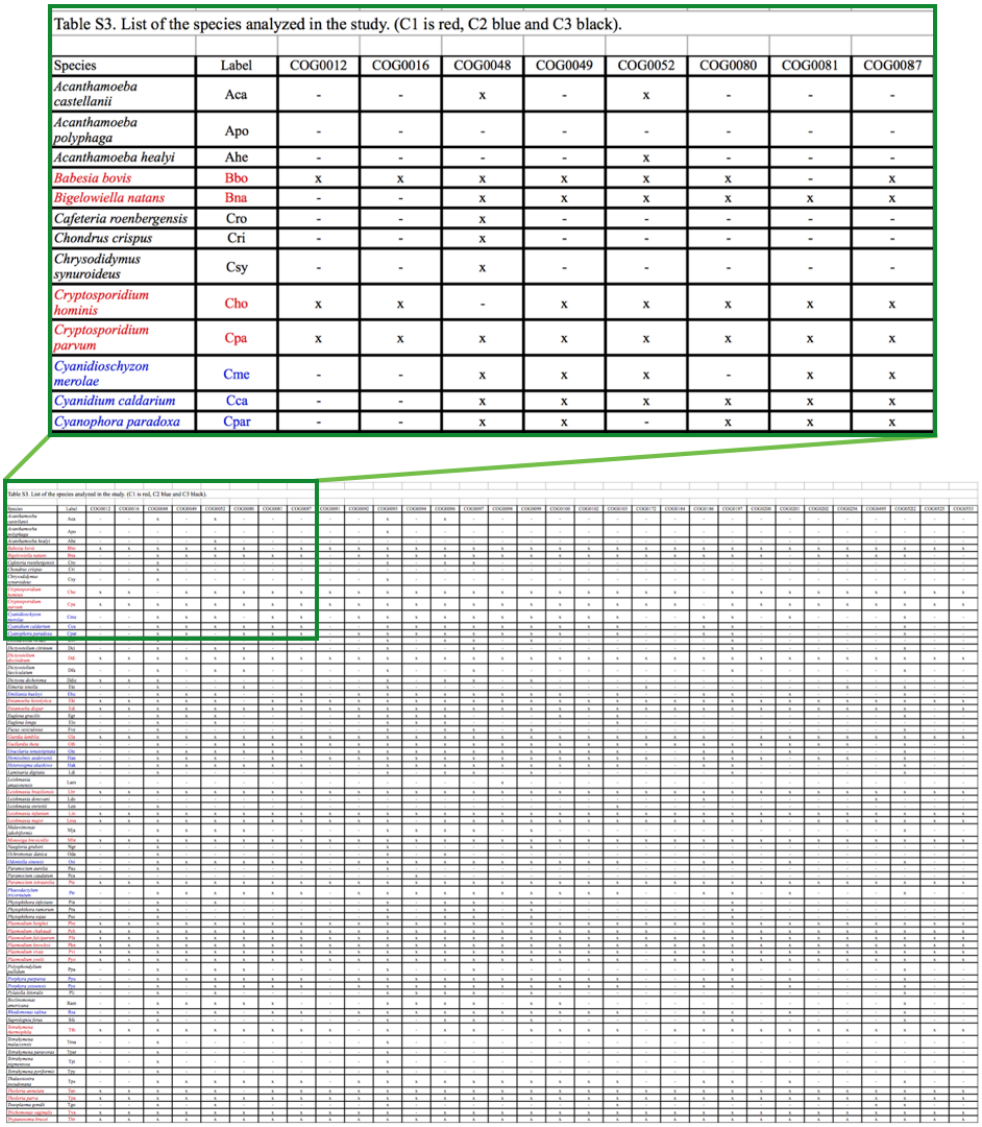

Table S3. List of the species analyzed in the study. (C1 is red, C2 blue and C3 black).

Species	Label	COG0012	COG0016	COG0048	COG0049	COG0052	COG0080	COG0081	COG0087
Acanthamoeba castellanii	Aca	-	-	x	-	x	-	-	-
Acanthamoeba polyphaga	Apo	-	-	-	-	-	-	-	-
Acanthamoeba healyi	Ahe	-	-	-	-	x	-	-	-
Babesia bovis	Bbo	x	x	x	x	x	x	-	x
Bigelowiella natans	Bna	-	-	x	x	x	x	x	x
Cafeteria roenbergensis	Cro	-	-	x	-	-	-	-	-
Chondrus crispus	Cri	-	-	x	-	-	-	-	-
Chrysodidymus synuroideus	Csy	-	-	x	-	-	-	-	-
Cryptosporidium hominis	Cho	x	x	-	x	x	x	x	x
Cryptosporidium parvum	Cpa	x	x	x	x	x	x	x	x
Cyanidioschyzon merolae	Cme	-	-	x	x	x	-	x	x
Cyanidium caldarium	Cca	-	-	x	x	x	x	x	x
Cyanophora paradoxa	Cpar	-	-	x	x	-	x	x	x

Figure 30 – Manual compilation strongly based on fact-checking.

Unlike Experiment 1, Experiment 2 does not require the use of KBC systems, thus making the overhead of work and complexity in the stack of technologies even leaner. Considering the case captured by Figure 30, from (Ocaña and Dávila 2011), the development of a Jupyter Notebook that automates the grounding of knowledge in a parameterizable fashion is all alone less laborious than the manual process. Additional benefits comprise: making the composition less error-prone, enhancing its reproducibility, allowing for reuse through parameterization, providing extended analytical capacity (as seen in the interactive visualizations), preserving the high-level rationales behind domain-specific decisions, as depicted by Figure 31, and allowing for re-execution of the composition workflow at any point in the future, keeping up with

relevant advances in the domain embodied as changes in the underlying dataset. Chapter 4 addressed these latter benefits (i.e., preserving the high-level rationales and allowing for re-execution), as Jupyter Notebooks fall short on the tracking of users' sessions, for instance, when it comes to tracking interactions made by scientists through GUI components.

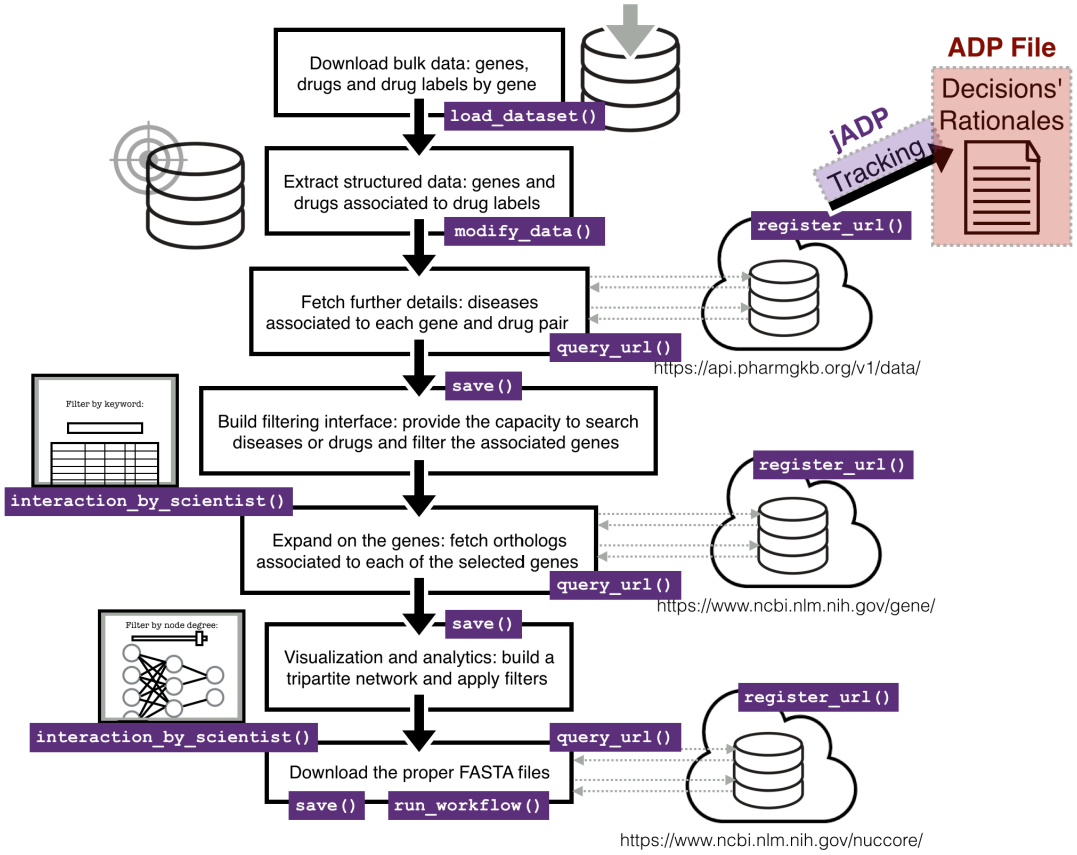

Figure 31 – jADP's operations in the context of Experiment 2.

Based on the results of Experiment 2, it is possible to conclude that:

● In comparison to previous experiences in the domain (Ocaña and Dávila 2011), we conclude that the development of a Jupyter Notebook to query all the KBs programmatically reduces the amount of human hours needed for the composition phase. According to domain expert's feedback, analyzing the results of hundreds of web searches and manually managing them in local spreadsheets is an extremely laborious task. This feedback also highlights the cost to recover the rationales behind all decisions made in the composition phase when returning to an experimental setting after several months or even years. So, with the support of the proposed jADP library, we believe

that an interactive computing environment is key to bring existing KBs to the loop. Consequently, it helps to reduce the amount of human hours needed when composing an experiment or recovering its underlying decisions at any given point in the future.

● With a Jupyter Notebook and all annotations in place, Experiment 2 can be re-executed at any given point in the future. Alternatively, the re-execution could be triggered after a certain amount of changes in the underlying KB data. The fact that PharmGKB more than quadrupled its coverage of human genes in 16 years (i.e., since its launch in 2001) (Hewett *et al.* 2002) shows how online KBs are live. This characteristic indicates that the automatic re-execution may be an important feature of the proposed approach.

● Considering the design choices of Jupyter Notebooks (Chamberlain and Schommer 2014), they can be deployed in virtualized operating systems with manageable access control – either for the users of Jupyter Notebooks and regarding applications' access to the external world –, such as Linux containers (Merkel 2014). These features are important considering either the policies of some online KBs and the architecture of several HPC infrastructures.

Chapter 6 – Conclusions and Future Work

Scientific experimentation is increasingly relying on data-intensive applications that are executed in HPC environments in order to reach new findings. A consequence of this trend is the increasing scale and complexity of domain-related facts that are valuable to domain experts while they compose, execute and analyze their experiments. For this reason, grounding scientific experiments in the most comprehensive and up-to-date domain knowledge is becoming a very important topic of research, with several known challenges.

In this dissertation, we studied how to bring domain knowledge to the loop of scientific experiments. Our approach is built on top of state-of-the-art technologies from the KBC and interactive computing communities (Ré *et al.* 2014)(Kluyver *et al.* 2016), while motivated and inspired by the state-of-the-art technologies from the workflow and data provenance communities (Mattoso *et al.* 2015)(Dias *et al.* 2015)(Murta *et al.* 2014)(Ogasawara *et al.* 2013). We evaluated our proposed approach experimentally in light of two of the three phases of the scientific experiment lifecycle. In these two complementary settings, we studied how scientific KBs could support HIL with the aid of Fact-Checkers, further analyzing the associated benefits. These analyses consider the current strategies that domain experts use to manage relevant knowledge in their experiments, trying to cope with the difficulties of scale and complexity in domain knowledge. In our two experiments, we find that current strategies for performing fact-checking have a number of limitations that are likely to be overlooked.

Experiment 1 explored the large scale of knowledge when analyzing a very large results space. In the context of this first experiment, we tested whether KBC systems could bring value at a manageable cost (Bursztyn, Dias and Mattoso 2016). We compared the benefits enabled by a domain-specific KB to the manual analytical process and found strong evidence on the limitations of human fact-checking.

Contrastingly, Experiment 2 explored the large scale and the substantial complexity in the knowledge required from domain experts when they compose Pharmacogenetics experiments. We verified that existing online KBs play an important role on their manual workflow composition (Hewett *et al.* 2002)(Maglott 2005),

although the potential advantage of these online KBs is only partially used. With this in mind, we tested whether additional tools could facilitate the automation of the composition phase while tracking and storing domain experts' decisions. We identified and addressed research opportunities in a toolset for interactive computing. We compared the benefits enabled by the overall solution to the manual composition process and found substantial enhancements motivating the use of online KBs.

In a nutshell, we contributed by investigating: (i) how relevant knowledge could be technically brought to the loop, and (ii) how it could create real value for scientists. The two original experiments devised for this work were largely complementary and helped to unveil a tangible set of benefits concerning the analysis phase and the composition phase. At a first glance, Experiments 1 and 2 leave good starting points for further research on the execution phase as they were built on top of current state-of-the-art technologies from the KBC and interactive computing communities, in partnership with a domain expert (Experiment 2). In a more in-depth view, it is also possible to anticipate why Experiment 1 and Experiment 2 are potentially valuable for research on the execution phase.

In the context of Experiment 1, in chapter 3, we simulated the possibility of using the knowledge in the loop to perform a parameter sweep with varying resolution. We showed that, for a relatively short sweep using a range of 15 values, that sort of parameter sweep would cut out undesired parts of this range. The strategy we described and simulated would minimize processing costs at no cost of quality, as it would still converge to the desired value of *k*. Implementing these ideas using more complex ranges of values – possibly with a more complex voting dataset –, appears to be an interesting research opportunity derived from Experiment 1.

In the context of Experiment 2, having it attached to a SWfMS that supports dynamic steering, such as Chiron, could be of great value for research on the execution phase. Dias (2013) argued that scientists could dynamically change SciPhy's MSA software depending on their time constraints, considering that there are different alternatives for the MSA activity associated to substantially different execution times. Similarly, in Experiment 2, scientists could interact with the threshold bar during an execution in order to meet a time constraint. In the case of that threshold bar (as in

Figure 29), a lower threshold could prompt an excessively large execution, while a higher threshold could shorten the execution time by delimiting a smaller and more strict scope in the experiment's composition, without losing sight of the subject under study (i.e., the disease used as keyword in the first interaction).

As suggested in our previous work (Bursztyn, Dias and Mattoso 2016), DeepDive and Chiron could be explored as in an integrated architecture, both dependent on declarative software artifacts. (Dias *et al.* 2015) also suggests that Chiron's unique support to dynamic steering makes it a SWfMS of great value for future research on the execution phase, where provenance data could be binded to a KB as a means to ground experiments at runtime.

To conclude, a good motivation for pursuing such opportunities is that research on the execution phase has a more direct impact on the second benefit envisioned at the beginning of this work: "saving HPC resources, as it enables the workflow to converge to the right direction more quickly (in some cases, with very low dependency to human fact-checking)."

Bibliographic References

Aluísio, S., Pelizzoni, J., Marchi, A., de Oliveira, L., Manenti, R. and Marquiafável, V. (2003). An Account of the Challenge of Tagging a Reference Corpus for Brazilian Portuguese. *Computational Processing of the Portuguese Language*, 194-194.

BioPython - Download (2017). http://biopython.org/wiki/Download, [accessed on June 5].

Bird, S. (2006, July). NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions* (pp. 69-72). Association for Computational Linguistics.

Bronaugh, W.F.J. (2007). 'Human-in-the-loop' Simulation: the Right Tool for Port Design. In Port Technology International.

Bursztyn, V. S., Dias, J. and Mattoso, M. (2016). Workflows Científicos com Apoio de Bases de Conhecimento em Tempo Real. In Proceedings of the X Brazilian e-Science Workshop.

Chamberlain, R. and Schommer, J. (2014). Using Docker to support reproducible research. *DOI: https://doi. org/10.6084/m9. figshare*, *1101910*.

Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., et al. (2009). Biopython: Freely Available Python Tools for Computational Molecular Biology and Bioinformatics. *Bioinformatics*, *25*(11), 1422-1423.

Dados Abertos - Legislativo (2016). http://www2.camara.leg.br/transparencia/dados-abertos/dados-abertos-legislativo/dados-abertos-legislativo, [accessed on April 7].

Davidson, S. B. and Freire, J. (2008). Provenance and Scientific Workflows: Challenges and Opportunities. In Proceedings of the 2008 ACM SIGMOD

DeepDive - DeepDive Applications (2016). http://deepdive.stanford.edu/showcase/apps, [accessed on April 7].

DeepDive - Knowledge Base Construction (2016). http://deepdive.stanford.edu/kbc, [accessed on April 7].

DeepDive - Tutorial: Extracting Mentions of Spouses from the News (2016). http://deepdive.stanford.edu/example-spouse, [accessed on April 7].

DIAP - Departamento Intersindical de Assessoria Parlamentar (2016). http://www.diap.org.br/, [accessed on April 7].

Dias, J. F. (2013). *EXECUÇÃO INTERATIVA DE EXPERIMENTOS CIENTÍFICOS COMPUTACIONAIS EM LARGA ESCALA* (Doctoral dissertation, Universidade Federal do Rio de Janeiro).

Dias, J., Guerra, G., Rochinha, F., et al. (may 2015). Data-centric Iteration in Dynamic Workflows. Future Generation Computer Systems, v. 46, p. 114–126.

Dias, J., Ogasawara, E., Oliveira, D., et al. (2011). Supporting Dynamic Parameter Sweep in Adaptive and User-Steered Workflow. In WORKS '11. ACM.

Endert, A., Hossain, M. S., Ramakrishnan, N., North, C., Fiaux, P. and Andrews, C. (2014). The Human is the Loop: New Directions for Visual Analytics. *Journal of intelligent information systems*, *43*(3), 411-435.

Gadelha, L. (2006). SINAPAD Grid PKI. Slides. 1st Meeting of The Americas Grid Policy Management Authority.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., et al. (2003). The Evolution of Protégé: An Environment for Knowledge-based Systems Development. *International Journal of Human-computer studies*, *58*(1), 89-123.

Glaab, E., Baudot, A., Krasnogor, N., Schneider, R. and Valencia, A. (2012). EnrichNet: Network-based Gene Set Enrichment Analysis. *Bioinformatics*, *28*(18), i451-i457.

Gonçalves, B. and Porto, F. (2015). Managing Scientific Hypotheses as Data with Support for Predictive Analytics. *Computing in Science & Engineering*, *17*(5), 35-43.

Gonçalves, B., Silva, F. C. and Porto, F. (2014). Y-DB: A System for Data-Driven Hypothesis Management and Analytics. *arXiv preprint arXiv:1411.7419*.

Hewett, M., Oliver, D. E., Rubin, D. L., Easton, K. L., Stuart, J. M., Altman, R. B. and Klein, T. E. (2002). PharmGKB: The Pharmacogenetics Knowledge Base. *Nucleic acids research*, *30*(1), 163-165.

Jagadish, H. V., Gehrke, J., Labrinidis, A., et al. (1 jul 2014). Big Data and its Technical Challenges. Communications of the ACM, v. 57, n. 7, p. 86–94.

Jupyter and the Future of IPython (2017). https://ipython.org/, [accessed on June 5].

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., et al. (2016). Jupyter Notebooks—A Publishing Format for Reproducible Computational Workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 87.

Knox, C., Shrivastava, S., Stothard, P., Eisner, R. and Wishart, D. S. (2007). BioSpider: A Web Server for Automating Metabolome Annotations. In *Pacific Symposium on Biocomputing* (Vol. 12, pp. 145-156).

Liu, Y., Li, Z., Xiong, H., Gao, X. and Wu, J. (2010, December). Understanding of Internal Clustering Validation Measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (pp. 911-916). IEEE.

MacQueen, J. (1967, June). Some Methods for Classification and Analysis of Multivariate Observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 1, No. 14, pp. 281-297).

Maglott, D., Ostell, J., Pruitt, K. D. and Tatusova, T. (2005). Entrez Gene: Gene-centered Information at NCBI. *Nucleic acids research*, *33*(suppl 1), D54-D58.

Manning, C. D., Raghavan, P. and Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

Masse, M. (2011). *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc.

Mattoso, M., Dias, J., Ocaña, K. A. C. S., et al. (may 2015). Dynamic Steering of HPC Scientific Workflows: A survey. Future Generation Computer Systems, v. 46, p. 100–113.

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Ogasawara, E., Oliveira, D., et al. (2010). Towards Supporting the Life Cycle of Large Scale Scientific Experiments. *International Journal of Business Process Integration and Management*, *5*(1), 79-92.

Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*, *2014*(239), 2.

Murta, L., Braganholo, V., Chirigati, F., Koop, D. and Freire, J. (2014, June). noWorkflow: Capturing and Analyzing Provenance of Scripts. In *International Provenance and Annotation Workshop* (pp. 71-83). Springer International Publishing.

Navarro, G. (mar 2001). A Guided Tour to Approximate String Matching. ACM Computing Surveys, v.33, n. 1, p. p 31–88.

Niu, F., Zhang, C., Ré, C. and Shavlik, J. W. (2012). DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference. *VLDS*, *12*, 25-28.

Ocaña, K. A. and Dávila, A. M. (2011). Phylogenomics-based reconstruction of protozoan species tree. *Evolutionary bioinformatics online*, *7*, 107.

Ocaña, K., de Oliveira, D., Ogasawara, E., Dávila, A., Lima, A. and Mattoso, M. (2011). SciPhy: A Cloud-based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes. *Advances in Bioinformatics and Computational Biology*, 66-70.

Ogasawara, E., Dias, J., Silva, V., et al. (2013). Chiron: A Parallel Engine for Algebraic Scientific Workflows. Concurrency and Computation, v. 25, n. 16, p. 2327–2341.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825-2830.

Pimentel, J. F. N., Braganholo, V., Murta, L. and Freire, J. (2015, July). Collecting and Analyzing Provenance on Interactive Notebooks: When IPython Meets noWorkflow. In *Workshop on the Theory and Practice of Provenance (TaPP), Edinburgh, Scotland* (pp. 155-167).

Pirolli, P. and Card, S. (2005, May). The Sensemaking Process and Leverage Points for Analyst Technology as Identified Through Cognitive Task Analysis. In *Proceedings of international conference on intelligence analysis* (Vol. 5, pp. 2-4).

Project Jupyter (2017). http://jupyter.org/, [accessed on June 5].

Ré, C., Sadeghian, A. A., Shan, Z., et al. (23 jul 2014). Feature Engineering for Knowledge Base Construction. arXiv:1407.6439 [cs].

Shin, J., Wu, S., Wang, F., De Sa, C., Zhang, C. and Ré, C. (2015). Incremental Knowledge Base Construction Using DeepDive. *Proceedings of the VLDB Endowment*, *8*(11), 1310-1321.

Travassos, G. H. and Barros, M. O. (2003, September). Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. In *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering* (pp. 117-130).

Waas, F. M. (2009). Beyond Conventional Data Warehousing - Massively Parallel Data Processing with Greenplum Database. *Business Intelligence for the Real-Time Enterprise*, 89-96.

Witten, I. H., Frank, E., Hall, M. A. and Pal, C. J. (2016). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

Yu, W., Clyne, M., Khoury, M. J. and Gwinn, M. (2009). Phenopedia and Genopedia: Disease-centered and Gene-centered Views of the Evolving Knowledge of Human Genetic Associations. *Bioinformatics*, *26*(1), 145-146.