Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate


Dottorato di Ricerca in Informatica

# DATA QUALITY EVALUATION THROUGH DATA QUALITY RULES AND DATA PROVENANCE

Antonella Zanzi

Advisor: Prof. Alberto Trombetta

**Abstract**

*The application and exploitation of large amounts of data play an ever-increasing role in today's research, government, and economy. Data understanding and decision making heavily rely on high quality data; therefore, in many different contexts, it is important to assess the quality of a dataset in order to determine if it is suitable to be used for a specific purpose. Moreover, as the access to and the exchange of datasets have become easier and more frequent, and as scientists increasingly use the World Wide Web to share scientific data, there is a growing need to know the provenance of a dataset (i.e., information about the processes and data sources that lead to its creation) in order to evaluate its trustworthiness. In this work, data quality rules and data provenance are used to evaluate the quality of datasets. Concerning the first topic, the applied solution consists in the identification of types of data constraints that can be useful as data quality rules and in the development of a software tool to evaluate a dataset on the basis of a set of rules expressed in the XML markup language. We selected some of the data constraints and dependencies already considered in the data quality field, but we also used order dependencies and existence constraints as quality rules. In addition, we developed some algorithms to discover the types of dependencies used in the tool. To deal with the provenance of data, the Open Provenance Model (OPM) was adopted, an experimental query language for querying OPM graphs stored in a relational database was implemented, and an approach to design OPM graphs was proposed.*

# Contents

CHAPTER 1

# Introduction

Data collection has become a normal function of institutions and organizations, and data errors can creep in at every step of the process from the initial acquisition to the storage in a repository. In particular, data acquisition from the field (e.g., through measurements or surveys) and data entry in digital repositories are inherently prone to errors.

The awareness of the critical importance of data quality issues has grown rapidly in the last decades in scientific, industrial, and governmental contexts. Data understanding and decision making heavily rely on high quality data; for this reason, it is important to be able to obtain such kind of data or, at least, to evaluate the quality of a dataset to judge if it is suitable for a specific purpose. It is often necessary to make decisions based on incomplete data; however, analyzing a topic or an issue relying on data of low quality, without having knowledge of it, can lead to wrong analysis results and decisions. Low quality data (sometime called "dirty data" [Rahm and Do, 2000]) include missing data, erroneous data, duplicates, inconsistent data, out-of-date data, and undocumented data.

For new archives or data collections, the prevention is obviously a fundamental aspect; still, issues can arise in the data management process when data are derived from other data or when a dataset results from the integration of multiple data sources. In the data warehouse field, for example, data quality issues can arise when different datasets are merged storing together inconsistent information or when records referring to the same entities are present in more than one data source with different values. Moreover, it might be necessary to deal with datasets received from external sources without having any control on the original acquisition process.

For existing datasets an option is to attempt to cleanse data. Data cleaning is not a simple task; it is highly context dependent, and – in many cases – data can be cleaned effectively only with some human intervention since fully automated

cleaning procedures could lead to a loss of useful information.

The discovery of incorrect data, which is the first necessary step in data cleaning, is – in most cases – challenging; moreover, even when the presence of errors is recognized, it is not always feasible to trace back the correct values (e.g., detecting inconsistencies among data may not be sufficient to determine which record is at fault), or it is not always possible to correct the data (e.g., changes in the original dataset are not allowed).

Another relevant aspect related to the quality of a dataset is its fitness for the intended purpose, which is one of the definitions provided for data quality, for example, in [Juran, 1964] "Data are of high quality if they are fit for their intended use in operations, decision making, and planning". It may occur that a dataset containing correct data is not useful in a specific context, for example, because a different level of detail is requested or for insufficient data coverage. In these cases, efforts need to be devoted to assess the quality level of datasets in order to evaluate their fitness for the intended purpose.

In this context, the concept of data provenance can be useful; especially when in a dataset changes on data are not allowed, it is important to be aware of the history of the dataset (i.e., its origin and modifications) to judge its reliability.

In the present work, we focus on the evaluation of the quality of a dataset using data quality rules and the concept of data provenance. Concerning the first topic, the applied solution consists in the identification of types of constraints that can be useful as data quality rules and in the development of a software tool to evaluate a dataset on the basis of a set of rules expressed in the XML markup language. As rule types, we selected some of the data constraints and dependencies already considered in the data quality field, but we also used order dependencies and existence constraints. To deal with information provenance, we adopted the Open Provenance Model (OPM), we implemented an experimental query language for querying OPM graphs stored in a relational database, and we proposed a method to approach the design of OPM graphs.

As a case study, we used some datasets collected by the Joint Research Centre of the European Commission, the Commission's in-house science service having the mission to provide European Union policies with independent, scientific, and technical support. The case study refers to real datasets in a context where low quality data limit the accuracy of the analysis results and, consequently, the significance of the provided policy advice. The case study was used to develop and test the experimented solutions, which are anyway generally applicable to other contexts.

## 1.1 Case study

In the European Union (EU) the fisheries sector is managed through the Common Fisheries Policy[1] (CFP).

Since the CFP was first established in 1983, scientific advice has increasingly become part of the fisheries management decision-making process. The role of the scientific advice was officially attested by the Commission Decision [European Commission, 31 August 2005] that established the Scientific, Technical and Economic Committee for Fisheries (STECF), a scientific advisory body to be "*consulted at regular intervals on matters pertaining to the conservation and management of living aquatic resources, including biological, economic, environmental, social and technical considerations*". The European Commission is thus required by regulation to take into account the advice from this committee when presenting proposals on fisheries management.

The are several impediments to the rational management of marine resources, one of them is inadequate data. Fisheries management decisions are often based on population models, but the models need high quality data to be effective. As stated in [Chen, 2003], the quality of fisheries data has great impact on the quality of stock assessment[2] and, consequently, on fisheries management.

The term *fisheries data* generally refers to data that may be useful in the management of a fishery. Such data usually include biological information about the exploited fish, economic information about the fishermen, and information about the environmental conditions that affect the productivity of the species.

Fisheries data are collected from different sources and can be categorized as fishery-dependent or fishery-independent data [Cooper, 2006]. Fishery-dependent data derive from the fishing process itself, while most of the fishery-independent data come from research surveys conducted by scientists.

The most common sources of fishery-dependent data are landings records and port samples. In some commercial fisheries, fishermen keep their own records, called logbooks or vessel trip reports, collecting data on the time and place of fishing, the effort expended, and the catch by species. Furthermore, onboard observers, who sometimes accompany fishing vessels, can provide information on fishing activities that are not always reported in logbooks, such as effects of fishing activities on

[1] http://ec.europa.eu/fisheries/cfp

[2] Stock assessment describes the past and current status of a fish stock; in addition, it attempts to make predictions about how the stock will respond to current and future management options [Cooper, 2006].

protected species and the extent and fate of bycatch[3] and discarding[4].

Fisheries data have different uses and many users, including strategic planning by industry and stock assessment by scientists. Data adequacy can be evaluated only in the context of the purposes for which they are used. For each use, data have to satisfy a set of requirements, including accuracy, coverage or completeness, level of detail, timeliness, accessibility to users, and credibility of both the data collection process and the management process that transforms and uses the data.

In order to allow the creation of a pan-European dataset to be used for policy advice, the Commission Regulation No 1639/2001 [European Commission, 17 August 2001] established the Data Collection Regulation (DCR), a Community framework for the collection of data in the fisheries sector; afterwards, the Commission Regulation No 665/2008 [European Commission, 15 July 2008] modified the DCR establishing the new Data Collection Framework (DCF) for the collection, management, and use of data in the fisheries sector with the aim to support the scientific advice regarding the CFP.

The new framework implements a routine and systematic collection of the basic data needed for scientific analyses and establishes rules and procedures to make these data available to the scientific community.

All the EU Member States that are involved in fisheries or aquaculture activities or in fish processing industry sector have to fulfill the DCF obligations. In particular, the DCF requires Member States to collect data on biological and economic aspects of many European fisheries and related fisheries sectors providing access to these data for fisheries management advice, scientific publication, public debate, and stakeholder participation in policy development.

The fisheries data collected under the DCF are classified as follows:

- Economic data: employment, expenditure, capital and investments, fishing rights, and direct subsidies;
- Biological data: length and age distribution, maturity data by age and length, sex ratio by age and length, discards, and discards length distribution;
- Effort data: days at sea and energy consumption;
- Transversal data: capacity, landings, and prices;

[3]Bycatch refers to fish or other species of animals caught unintentionally.

[4]Discards are the portions of a catch which are not retained on board and are returned, often dead or dying, to the sea.

– Data from surveys (i.e., sampling at sea) about demersal species and small pelagic species[5].

The Joint Research Centre (JRC), on behalf of the Directorate-General for Maritime Affairs and Fisheries (DG-MARE) of the European Commission, collects and maintains the fisheries data reported by EU Member States under the DCF. The datasets gathered by JRC are first checked for compliance and quality, then they are placed at disposal of teams of independent experts who, participating in STECF working groups, are in charge of the scientific advice.

Being the scientific advice based on the results of the analyses performed by the experts, the assessment of the quality of the data they use is an important and critical task. In fact, even if in the DCF regulation it is stated that "*Member States shall be responsible for the quality and completeness of the primary data collected* [...] *and for detailed and aggregated data derived* [...]", there are several issues in the quality of the data provided by Member States.

In a recent document about fishing opportunities for 2011 produced by the European Commission, it is reported that the scientific advice about overfishing is missing for about two-thirds of the total allowable catches[6]. In most cases, this is because of missing information on catches, incomplete surveys, or poor sampling, though there are cases where the underlying biological issues present difficult scientific challenges [European Commission, 2011].

The data quality checks carried out at JRC is concretely helping Member States in assessing the quality of the data they provide and also in improving the quality of their data management process. Detailed information about the DCF data collection activities performed by JRC can be found in a dedicated Web site[7]; in addition, the results of the data analyses performed by the experts are published on the STECF Web site[8].

### 1.1.1 An IP-MAP for the case study

To illustrate the processes involved in the fisheries data submission from Member States, with particular emphasis on the activities addressing data quality verification and improvement, we use an Information Product Map (IP-MAP), which is a graphical language for the description of the information production processes

[5]Demersal marine species live on or near the bottom of the sea, whereas pelagic marine species live near the surface of the sea.

[6]Total allowable catches are the catch limits for most significant commercial fish stocks and are decided every year by the Council of Ministers of the EU.

[7]https://datacollection.jrc.ec.europa.eu

[8]https://stecf.jrc.ec.europa.eu

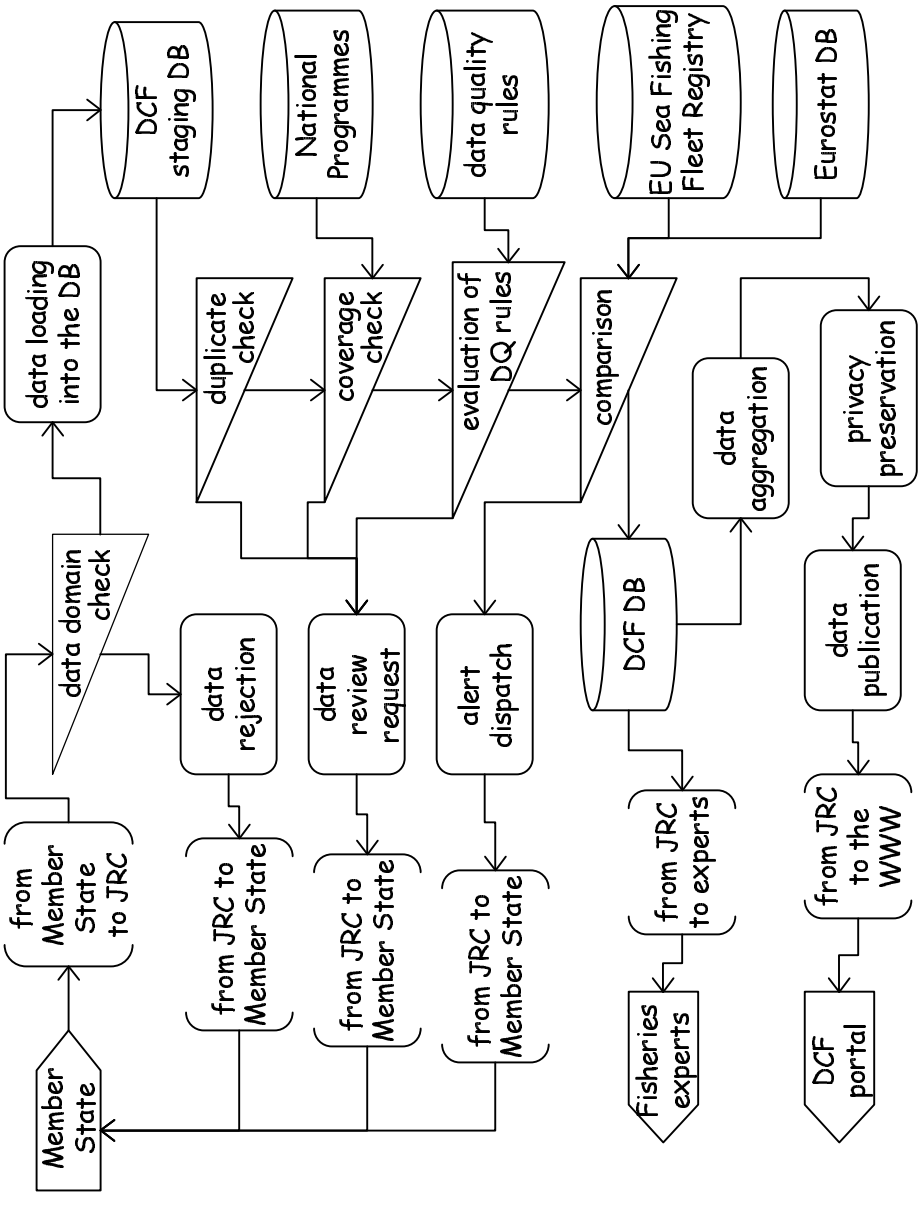



**Figure 1.1:** An IP-MAP for the case study

[Shankaranarayan et al., 2000] (details about the constructs used to build IP-MAPs can be found in appendix C).

The IP-MAP in figure 1.1 refers to one Member State and one dataset; the same process is repeated for all the Member States and all the datasets involved in the DCF. The main tasks related to data quality are shown in the diagram: domain checks, duplicate discovery, evaluation of data quality rules, and comparisons with other data sources.

During the data submission procedure, some preliminary checks are carried out and Member States are informed about issues in the format of their data. At the end of the submission procedure, other checks are performed on the data stored in the staging database.

First of all, duplicate records and records with different values referring to the same entities (e.g., the same fleet segment) are identified (i.e., duplicate detection and record linkage procedures are applied).

Afterwards, data are evaluated against a set of rules which are mainly provided by experts in the fisheries sector; in the diagram, the data quality block labeled "evaluation of DQ rules" refers to this step.

Furthermore, for each country, coverage checks are carried out to ensure that all the necessary data for each fleet segment have been submitted. To establish how many and which fleet segments should be reported by each country, the National Programmes of each Member State are consulted.

Finally, external data sources are also used to improve the consistency checks on the received data: the Eurostat database[9] and the EU Sea Fishing Fleet Register[10]. Eurostat, the statistical office of the European Union providing statistics at European level, collects also some data about fisheries (i.e., catches and landings data aggregated at national level) and aquaculture activities. The Sea Fishing Fleet Register is a repository where all the fishing vessels flying the flag of a Member State have to be registered in accordance with the Community legislation; in particular, capacity data (i.e., the number of vessels, the kilowatts, and the gross tonnages) are requested for each fleet segment for the entire national fleet. Data from these two archives are used to assess the consistency, at national level, of some of the data collected under the DCF.

[9] Eurostat data are accessible online at http://epp.eurostat.ec.europa.eu

[10] Fishing Fleet Register data are accessible online at http://ec.europa.eu/fisheries/fleet/index.cfm

## 1.2 Work content

The aim of the work is to explore, apply, and – if necessary – develop approaches and tools to evaluate and improve the quality of datasets in a scientific context. The motivation of the work is a real need to evaluate and improve the quality of data in the context of the EU Data Collection Framework for fisheries data introduced in the previous paragraph, but the objective is to propose solutions generally applicable regardless of the context.

After a literature survey and a comparison of the available techniques and tools, we decided to utilize some already existing robust approaches (e.g., database profiling, exploratory data analysis, and duplicate detection) to deal with some of the issues related to data quality; whereas the development effort mainly addressed two topics that are relevant in the context of data quality evaluation and for which there are not yet well-established approaches and tools: the use of quality rules and provenance information.

The concept of rules is widely used in data management and data quality fields: integrity constraints, functional dependencies, and business rules are types of rules normally enforced on datasets. However, rules can play an important role also in the data quality evaluation field. In this context, rules are not constraints to be enforced on a database but are used to determine the quality profile of a dataset. Data quality rules can verify not only the correctness of data but also their suitability for a purpose. Thus, more than one set of rules can be created for a dataset in order to verify its fitness for different purposes. Furthermore, the concept of data quality rules can be used to quantify data quality using the results of the evaluation of the rules in order to compute data quality metrics.

We adopted an approach in which quality constraints on data are expressed in form of rules, and we developed a software tool for the evaluation of a dataset on the basis of a set of rules. As data quality rule types to be used in our tool, first of all we selected some types of data constraints and dependencies already proposed in data quality works, for example, association rules, functional dependencies, and conditional functional dependencies (the latest being functional dependencies holding on a subset of the relation instance and recently introduced in the data cleaning context [Bohannon et al., 2007]). In addition, we considered *order dependencies* [Ginsburg and Hull, 1983] and *existence constraints* [Atzeni and Morfuni, 1986] because they can be useful in the context of data quality evaluation, even if, to our knowledge, they have not yet been used in the data quality field.

The data collected under the DCF are not well documented with provenance information. Some information can be found in the National Programmes (e.g.,

the description of data sampling schemes and methodological choices), but further information can be useful to allow experts and other users to evaluate the reliability of the data provided by Member States.

With the aim to deal with this shortcoming, we explored the possibility to accompany the data published on the DCF Web site with provenance information, and we experimented the adoption of a model for data provenance recently proposed in literature, the Open Provenance Model.

The rest of the thesis is organized as follows:

- Chapter 2 discusses the definition of data quality, introduces some classifications of data quality issues, and summarizes the current approaches used to address them.

- Chapter 3 introduces various types of constraints and dependencies that can be used as data quality rules.

- In chapter 4, we describe the solution adopted to verify if a dataset complies with a set of rules expressed in the XML markup language, and we introduce the metrics used in the context of the used case study.

- Chapter 5 presents the algorithms implemented to discover the data dependencies called Conditional Functional Dependencies (CFD) and CFD$^{\mathrm{p}}$ (which are CFD with built-in predicates). These dependencies, which are extensions of traditional functional dependencies, have been recently proposed in literature in the data quality field.

- Chapter 6 introduces the concept of data provenance and describes the Open Provenance Model (OPM), a model of provenance that aims to promote the exchange of provenance information among heterogeneous systems.

- In chapter 7, we propose a method for the design of OPM graphs. The method is composed of two phases: in the conceptual phase, concepts and their relationships are identified and mapped into OPM elements; the physical phase refers to the serialization of the OPM graph into the corresponding XML structure with the validation of the generated XML document.

- Chapter 8 describes the approach used to store and query an OPM graph in a relational database: a relational schema for OPM was designed and queries are performed on it.

CHAPTER 2

# Data Quality

## 2.1 Data Quality definition

Considering the quality of data, the first thoughts that come to mind are the correctness of data and the need to have error-free data. Certainly, this is a very important aspect of the problem, but it is not the only one determining the quality level of data. For this reason, different definitions for the concept of data quality have been proposed, and still a formal one is missing. In the following, some of the several definitions proposed in literature are introduced:

- Juran [1964] focuses on the use of data: "Data are of high quality if they are fit for their intended use in operations, decision making, and planning";
- according to Orr [1998] data quality "is the measure of the agreement between the data views presented by an information system and that same data in the real world";
- Shanks and Corbitt [1999], following the Juran's point of view, adopt the definition of quality as "fitness for purpose";
- Redman [2001] highlights data characteristics: "Data to be fit for use must be accessible, accurate, timely, complete, consistent with other sources, relevant, comprehensive, provide a proper level of detail, be easy to read and easy to interpret".

In the last reported definition, data quality is interpreted as a combination of properties or characteristics, sometime called dimensions [Scannapieco and Catarci, 2002] to emphasize the fact that they may be quantified (even if in reality it is not often straightforward to measure data quality characteristics); examples of these properties are: accuracy, completeness, consistency, relevance, and accessibility.

A first survey of the proposed sets of dimensions characterizing data quality was published in 1995 [Wang et al., 1995]. Since then, other proposals have been presented, such as in [Naumann, 2002] and [Bovee et al., 2003]. Moreover, several international organizations adopted their own set of characteristics defining data quality (some examples can be found in [Bergdahl et al., 2007, OECD, 2008, European Commission, 31 March 2009]).

Even if there is not a common agreement on the set of properties to be used to characterize data quality, some of them are considered particularly important (e.g., accuracy, consistency, and completeness) and always cited in data quality works; still, for some of them (e.g., reliability, believability, credibility, and accessibility) the assigned meaning can be slightly different.

The following list shows the most mentioned data quality properties both in literature works and in reports published by institutions providing statistical data:

**Accuracy** is considered the most important characteristic of data quality. It refers to the lack of errors and it is defined as the degree of correctness of a recorded value when compared with the real-world value. Inaccurate data can be either incorrect or out-of-date data.

**Consistency** means that the representation of the data value is the same in all cases. It implies that there is no redundancy and that, in a database context, referential integrity is enforced. Conflicting data are considered inconsistent.

**Completeness** is defined as the degree of presence of data in a given collection.

**Validity** means that an attribute is valid if its value is within a predefined value domain.

**Believability** is the extent to which data are accepted or regarded as true and credible.

**Credibility** refers to the credibility of the data source.

**Relevance** refers to the applicability of data in a particular context; it is the degree to which data meet current and potential user needs.

**Timeliness** of information reflects the length of time between data availability and the event or phenomenon they describe; it implies that the recorded value is not out-of-date. Timeliness sometime is defined in terms of *currency*, how recent are data, and *volatility*, how long data remain valid.

**Accessibility** refers to the physical conditions under which users can obtain data.

## 2.2 Classification of Data Quality issues

In addition to the given definitions of data quality, it is interesting to consider more in detail which are the problems that affect in practice the quality of a dataset.

In [Oliveira et al., 2005b] it is stated that, to measure the coverage of a tool for the automatic or semi-automatic detection and correction of the problems that affect data, a complete taxonomy of quality problems should be used. In practice, beyond the lack of a standard definition for the data quality concept, there is no standard taxonomy of data quality issues; however, some classifications of the problems that affect data quality have been proposed in literature.

In [Oliveira et al., 2005b,a] data quality problems are divided in single-source and multi-source, then 26 and 9 primitive problems are respectively identified in these two groups.

Also Rahm and Do [2000] distinguish between single-source and multi-source data quality problems. Moreover, they divide the two groups into schema-related and instance-related problems, defining schema-related problems as those that can be addressed by improving the schema design, schema translation, and schema integration, and defining instance-related problems as errors and inconsistencies in the actual data contents that cannot be prevented at the schema level.

Kim et al. [2003] present a detailed taxonomy of data quality issues consisting of 33 primitive data quality problems. The taxonomy is based on the premise that data quality problems manifest in three different ways: missing data; not missing, but wrong data; and not missing and not wrong, but unusable data. Unusable data can occur when two or more databases are merged or when codifications are not consistently used. The taxonomy is a hierarchical decomposition of these three basic manifestations of data quality issues.

Müller and Freytag [2003] roughly classify data quality issues into syntactical, semantic and coverage anomalies. Syntactical anomalies describe characteristics concerning the format and values used to represent entities (e.g., lexical errors and domain format errors); semantic anomalies hinder the data collection from being a comprehensive and non-redundant representation of the world (e.g., duplicates and contradictions), and coverage anomalies decrease the amount of entities and entities properties from the world that is represented in the data collection (e.g., missing values). This classification is limited to data quality problems that occur in a single relation of a single source.

Finally, a more general typology of data quality issues is proposed in [Fürber and Hepp, 2005], where the quality problems are traced back to four basic types; namely, inconsistency, lack of comprehensibility, heterogeneity, and redundancy.

## 2.3 Approaches to Data Quality

Approaches to data quality can be divided in preventive, diagnostic, and corrective approaches.

Being data cleaning an expensive process and not always applicable, preventing the inclusion of dirty data in a repository is obviously an important step. Preventive approaches refer, for example, to the appropriate design of databases, to the use of integrity constraints, and to the development of data entry tools providing data check functionalities.

Prevention, however, is not always sufficient to avoid all the possible quality issues; thus, the discovery of incorrect data is an important task in the evaluation of data quality, and it is the first necessary step in data cleaning. Common diagnostic approaches are database profiling and exploratory data analysis.

Finally, data cleaning deals with detecting and removing errors and inconsistencies from data. Currently used corrective approaches comprise cleaning methods for attributes, duplicate detection techniques, and virtual repair methods.

For the variety of the techniques proposed to deal with low quality data, efforts have been recently devoted also to the definition of methodologies that can help in selecting, customizing, and applying data quality techniques.

The rest of the chapter briefly summarizes the approaches to data quality used in existing software tools or that have been presented in literature.

### 2.3.1 Diagnostic approaches

#### Database profiling

Data profiling focuses on the analysis of individual attributes. It derives information such as data type, length, value range, discrete values and their frequency, variance, uniqueness, occurrence of null values, and typical string pattern, providing an exact view of various quality characteristics of the attributes.

#### Exploratory data analysis

Exploratory Data Analysis (EDA), sometimes called Exploratory Data Mining [Dasu and Johnson, 2003], refers to activities that enclose the statistical evaluation of data values and the application of data mining algorithms in order to explore and understand data.

Data mining can help discover specific data patterns (e.g., relationships holding among several attributes) and rules to ensure that data do not violate the application

domain constraints. This is the focus of so-called descriptive data mining models including clustering, summarization, association discovery, and sequence discovery.

In many, if not in most cases, data can only be cleaned effectively with human involvement. EDA typically involves a human in the process of understanding properties of a dataset, including the identification and possible correction of errors. Therefore, there is typically an interaction between EDA methods and data visualization systems: data visualization is often used to make statistical properties of data (e.g., distributions, correlations, etc.) easily accessible to data analysts.

### 2.3.2 Corrective approaches

#### Cleaning methods for attributes

Data cleaning techniques for single attributes can be categorized by the data types that they target.

In this field, the issue of cleaning quantitative data (i.e., integers or floating point numbers) has been initially addressed. Statistical methods for outlier detection are the foundation of data cleaning techniques in this domain; an exhaustive survey of data cleaning methods used on quantitative attributes can be found in [Hellerstein, 2008].

Research efforts have been also devoted to categorical data, namely, names or codes used to assign data into categories or groups. One key problem in data cleaning for categorical data is the identification of synonyms. Another relevant issue is managing data entry errors (e.g., misspellings) that often arise with textual codes; currently, several techniques for handling misspellings, often adapted to specialized domains, are available.

Recent works have proposed the use of Functional Dependencies (FDs) for data cleaning purposes in relational databases. Data dependencies are normally used in the context of database design; however, representing domain knowledge, they can be useful also in the data quality field. For example, in [Pivert and Prade, 2010] the case in which dirtiness corresponds to the violation of FDs is considered.

Furthermore, in order to perform more specific data cleaning operations, extensions of FDs have been proposed. In particular, Conditional Functional Dependencies (CFDs), which are FDs holding on a subset of the instances of the original relation, have been recently proposed in [Cong et al., 2007] as a method for inconsistency detection and repairing. This approach is used, for example, in *Semandaq* [Fan et al., 2008a], a tool using CFDs for data cleaning purposes, where users can specify CFDs through the drag and drop functionality provided in the user interface. Another tool, called *Data Auditor*, is presented in [Golab et al., 2010] and

supports more types of constraints (i.e., CFDs, *conditional inclusion dependencies*, and *conditional sequential dependencies*) that are used to test data inconsistency and completeness.

### Duplicate detection and record linkage

The aim of these techniques is to match all the records relating to the same entity. Duplicate entries can be identical records or records referring to the same entity and containing different values.

The process of matching records stored in different databases but referring to the same entity is commonly called record linkage; other terms are also used, such as merge/purge problem, object identification, and entity uncertainty. Sometime, when applied on a single database, this process is called de-duplication.

Different techniques, such as knowledge-based methods, filtering, and regular expression matching, have been proposed to address this issue; a survey of the most used techniques can be found in [Elmagarmid et al., 2007].

### Querying inconsistent data

An approach to manage data inconsistencies is based on the concepts of repair and consistent query answer [Bertossi and Chomicki, 2003].

A repair of a database is another database – over the same schema – that is consistent and differs minimally from the original database. Research has focused on both materialized and virtual repairing, and different notions of repair have been proposed to capture the concept of minimal change in different ways [Chomicki, 2006].

Virtual repairing, which is usually called consistent query answering, does not change the database but rather returns query answers true in all repairs (i.e., consistent query answers) regardless of the way the database is fixed to remove constraint violations. Thus, consistent query answers can provide a conservative "lower bound" on the information contained in the database. With this aim, various methods for computing consistent query answers without explicitly computing all repairs have been developed [Chomicki, 2006].

### Data transformation methods

Data transformation methods are used to modify the schema and the instances of a database, mainly with integration purposes.

In particular, the Extract-Transform-Load process for data warehouse creation consists of steps that extract relevant data from the sources, transform them to the

target format, clean it, and then load them into the data warehouse. A survey on the technologies that can be used in the Extract-Transform-Load process can be found in [Vassiliadis, 2009].

## 2.4 Data Quality assessment

As summarized in [Lee et al., 2006], the main techniques that are currently used to assess the quality of a dataset are:

- The use of data quality surveys, which relies on users satisfaction as a measure of quality; a survey elicits the evaluation of data quality dimensions from stakeholders of the organization providing the data and the resulting assessment reflects the perceptions of the participants in the survey.
- The computation of data quality metrics or indicators, which can provide objective measurements of some quality characteristics.
- The application of data integrity analysis, which focuses on the conformance of data to integrity constraints specified in the database.

To assess the quality of data stored in a repository or produced by an organization, different strategies employing combinations of these techniques can be adopted.

Several proposals of combined techniques in form of methodologies, mainly addressed to business companies and organizations providing services to clients, can be found in literature and some of them will be mentioned in one of the following paragraphs.

### 2.4.1 Metrics for Data Quality

Assessment of data quality can be performed in relation to several data quality dimensions.

Data quality dimensions and their measurement are usually discussed independently from each other. In each context, after having determined which data quality dimensions are important or more useful, it is then necessary to define some variables referring to the chosen dimensions.

The following are examples of general metrics suggested in [Lee et al., 2006]:

- For the accuracy dimension:

$$\mathsf{Free-of-error\ rating} = 1 - \frac{\mathsf{Number\ of\ data\ units\ in\ error}}{\mathsf{Total\ number\ of\ data\ units}}$$

– For the completeness dimension:

$$\text{Completeness rating} = 1 - \frac{\text{Number of incomplete items}}{\text{Total number of items}}$$

– For the consistency dimension:

$$\text{Consistency rating} = 1 - \frac{\text{Number of instances violating specific consistency type}}{\text{Total number of consistency checks performed}}$$

– For the believability dimension:

$$\text{Believability} = min(\text{Believability of source}, \text{Believability based on age of data}, \text{Believability when compared to internal commonsense standard})$$

where each of the three used variables are rated on a scale from 0 to 1.

– For the timeliness dimension (reflecting the up-to-date data feature):

$$\text{Timeliness rating} = \left\{ max\left[1 - \frac{\text{Currency}}{\text{Volatility}}, 0\right] \right\}^{s}$$

where:

- currency = (delivery time - input time) + age;
- volatility refers to the length of time over which data remain valid;
- delivery time refers to the time at which data were delivered to the user;
- input time refers to the time at which data were received by the system;
- age refers to the age of data when were first received by the system;
- and the exponent value is task dependent (allowing one to control the sensitivity of the measure).

Many of the variables that can be measured are context dependent, for this reason, in order to be used in practice, the above proposed metrics need to be customized in the specific context.

For example, the metric suggested for the accuracy dimension, which is the most cited data quality characteristic, can be used only when wrong data are easily identifiable, which is not often the case. Accuracy can be defined as how closely information matches a real-life state; namely, the accuracy of an attribute value measures the distance between the attribute value stored in a data repository and

the corresponding value in the real world at the moment of measuring the quality of the attribute value. Accuracy is thus determined by means of a distance measure; to measure it exactly, it is necessary to know the attribute value as it is stored in the data source and also its actual value. In many cases, determining the latter is challenging; moreover, in some cases, it is not even possible (this is often the case for data in scientific contexts). Therefore, it could be useful to derive information on the quality of an attribute value without knowing its real-world counterpart. Görz and Kaiser [2012], for example, investigate how metrics for completeness, validity, and currency dimensions can be aggregated to derive an indicator for accuracy and propose an indicator function for accuracy based on the algebraic product of the measures for the completeness, validity, and currency dimensions.

## 2.5 Data Quality methodologies

A data quality methodology can be defined as a set of guidelines and techniques that, starting from the description of an application context, defines a rational process to assess and improve the quality of a dataset.

The Total Data Quality Management (TDQM) [Wang, 1998] was the first general methodology published in the data quality literature. The fundamental objective of this methodology is to extend the principles of Total Quality Management used in the field of product manufacturing to the data quality context. The process underlying the TDQM methodology considers four phases (iteratively executed, thus constituting a cycle) as necessary for managing an information product:

1. The definition phase includes the identification of data quality dimensions and related requirements.

2. The measurement phase produces quality metrics that provide feedback to data quality management and allow for the comparison of the effective quality with predefined quality requirements.

3. The analysis phase identifies the roots of quality problems and studies their relationships.

4. The improvement phase devises quality improvement activities.

TDQM proposes also a graphical model for the description of the information production processes, called IP-MAP [Shankaranarayan et al., 2000]. IP-MAP enables the specification of business processes by means of a conceptual map in which the activities corresponding to the data quality management process are properly addressed.

IP-MAP is the only language for information process modeling ever proposed and represents a de facto standard. A comparison among IP-MAP and other graphical modeling languages (e.g., workflow and data flow diagrams) can be found in [Shankaranarayan and Wang, 2007]. An example of IP-MAP for the case study was presented in the introduction of the present work; moreover, details about the constructs of the language can be found in appendix C.

Afterwards, other methodologies have been proposed, such as AIMQ (A Methodology for Information Quality Assessment) [Lee et al., 2002], DQA (Data Quality Assessment) [Pipino et al., 2002], and CDQ (Comprehensive methodology for Data Quality management) [Batini and Scannapieco, 2006].

The sequence of activities composing a data quality methodology can be generally grouped in three phases:

1. State reconstruction, which is aimed at collecting contextual information on organizational processes and services, data collections and related management procedures, quality issues and corresponding costs.

2. Assessment/measurement, which measures the quality of data collections along relevant quality dimensions. The term measurement is used to address the issue of measuring the value of a set of data quality dimensions, while the term assessment is generally used when such measurements are compared with reference values to enable a diagnosis of quality.

3. Improvement, which concerns the selection of steps, strategies, and techniques in order to reach new data quality targets.

A detailed comparison of thirteen methodologies (comprised those previously mentioned) can be found in [Batini et al., 2009], where the considered methodologies are compared along several perspectives, including the methodological steps, the techniques, the data quality dimensions, the types of data, and the types of information systems addressed by each methodology.

## 2.6 Data cleaning tools

Every software application that implements one or more techniques to approach one or more data quality issues can be considered a data quality tool, including in the group, for example, also data analysis and data profiling tools. However, only the tools aiming at correcting data anomalies are generally considered to be cleaning tools.

Common functionalities provided by the available data cleaning tools are cleaning of categorical attributes, outlier identification, duplicate detection, and record linkage. Some tools concentrate on a specific domain, such as cleaning name and address data, or on a specific cleaning phase, such as duplicate detection. At the contrary, Extract-Transform-Load (ETL) tools provide transformation and workflow capabilities covering a large part of the data transformation and cleaning process necessary in the construction of data warehouses.

Currently, there are several commercial tools available for data cleaning and most of them are ETL tools. Industrial systems for ETL are provided both by the major database vendors and by the individual ETL-targeted vendors. Popular tools include Oracle Warehouse Builder, Informatica PowerCenter, IBM Datastage, and Microsoft Integration Services.

In [Barateiro and Galhardas, 2005] a survey on data cleaning tools is presented, with a detailed feature comparison on tools both of academic and industrial origin; moreover, a survey on record linkage tools, both academic and commercial, can be found in [Gu et al., 2003].

Some open source tools are also available. Among the open source tools we are aware of, those that seem to have the most active communities are the following: Febrl, which is mainly a record linkage tool, SQL Power DQguru, Flamingo Package, and four ETL tools called Kettle, Talend Open Studio, CloverETL, and KETL.

Furthermore, data cleaning tools have also been developed by the research and academic community. Some of them (FraQL, IntelliClean, AJAX, Potter's Wheel, SmartClean, Arktos, HumMer/FuSem and TAILOR) will be introduced in the following subsection along with the previously mentioned open source tools.

### 2.6.1 Tools from the research community

#### FraQL

FraQL [Sattler and Schallehn, 2001] is a declarative language supporting the specification of a data cleaning process. The language is an extension of SQL based on an object-relational data model. It supports the specification of schema transformations as well as data transformations using user-defined functions. FraQL supports also the filling in of missing values, the elimination of invalid tuples, and the removal of outliers.

#### IntelliClean

IntelliClean [Lee et al., 2000] is a rule-based approach to data cleaning with the main focus on duplicate elimination. The proposed framework comprises three steps. In

the pre-processing stage syntactical errors are eliminated and the values are standardized in format. The processing stage represents the evaluation of cleaning rules that specify actions to be taken under certain circumstances. There are four different classes of rules: (1) duplicate identification rules specify the conditions under which tuples are classified as duplicates; (2) merge/purge rules specify how duplicates are to be handled; (3) update rules specify the way data is to be updated in a particular situation and can also be used to specify how missing values ought to be filled in; (4) alert rules specify conditions under which the user is notified about an event occurrence or a constraint violation. During the first two steps of the data cleaning process the actions taken are logged providing documentation of the performed operations. In the human verification and validation step, these logs are investigated to verify and possibly correct the performed actions.

### AJAX

AJAX [Galhardas et al., 2000] major concern is transforming existing data from one or more data collections into a target schema eliminating duplicates within the process. For this purpose, a declarative language for expressing transformations to be performed on tables is defined. AJAX provides five transformation operators: mapping, view, matching, clustering, and merging. (1) Mapping is used to split one table into several tables. (2) View corresponds to an SQL query augmented by integrity constraints over its result; integrity constraints can generate exceptions that correspond to specific events in the process. (3) Matching computes an approximate join between two relations using, instead of the equality operator of SQL, a distance function to decide which pairs of values need to be joined. (4) Clustering takes a single input relation and returns a single output relation that groups the records of the input relation into a set of clusters; clusters can be calculated on the basis of the usual SQL `group by` operator or by means of a distance function. (5) Merging partitions an input relation according to various grouping attributes and collapses each partition into a single tuple using an arbitrary aggregation function.

### Potter's Wheel

Potter's Wheel [Raman and Hellerstein, 2001] is an interactive data cleaning tool that integrates data transformation and error detection using a spreadsheet-like interface.

The tool provides the following operations, called transforms, to support common schema transformations without explicit programming: value translations, applying a function to every value in a column; one-to-one mappings, transforming individual

rows; and many-to-many mappings of rows. The effects of the performed operations are shown on tuples visible on the screen. The anomalies handled by this approach are syntax errors and irregularities. Users can also specify the required transformation through examples, and the tool produces the function that best matches the provided examples by using algorithms based on the identification of regular expressions. In addition, the tool allows users to define custom domains and corresponding algorithms to enforce domain constraints.

#### SmartClean

SmartClean [Oliveira et al., 2009] is a data cleaning tool in which both detection and correction operations are specified through a declarative language inspired by the SQL language. The user does not need to specify the execution order of data cleaning operations, the sequence is automatically established by the tool. The tool supports the manipulation of data quality problems at different levels of granularity (i.e., tuple, relation, and multiple relations).

#### Arktos

Arktos [Vassiliadis et al., 2001] is a framework modeling and executing the ETL process for data warehouse creation. In this tool, data cleaning is considered as an integral part of the ETL process; for this reason a meta-model is specified allowing the modeling of the complete ETL process. The single steps, which perform cleaning operations, within the process are called activities. Each activity is linked to input and output relations, and the logic performed by an activity is declaratively described by an SQL statement. Each statement is associated with a particular error type and a policy that specifies the behavior (i.e., the action to be performed) in case of error occurrence. Six types of errors can be specified: primary key violation, uniqueness violation and reference violation are special cases of integrity constraint violations; the error type null existence is concerned with missing values; the remaining error types are domain mismatch and format mismatch referring to lexical and domain format errors.

#### HumMer/FuSem

Humboldt Merger (HumMer) [Bilke et al., 2005] and FuSem [Bleiholder et al., 2007] (which is an extension of HumMer) allow ad hoc declarative fusion of data from heterogeneous sources using an extension of the SQL syntax to support data fusion operations. Guided by a query against multiple tables, HumMer proceeds in three

automated steps. First, instance-based schema matching bridges schematic heterogeneity of the tables by aligning corresponding attributes. Next, duplicate detection techniques find multiple representations of identical real-world objects. Finally, the data fusion and conflict resolution step merges duplicates into a single and consistent representation. HumMer provides a subset of SQL as a query language, which consists of select-project-join queries, and allows sorting, grouping, and aggregation. In addition, it supports the Fuse By statement [Bleiholder and Naumann, 2005] allowing both the alignment of different relations and the identification of tuples representing the same real-world object.

#### TAILOR

TAILOR [Elfeky et al., 2002] is a record matching toolbox that allows users to apply different duplicate detection methods on a dataset. TAILOR provides four main functionalities: searching method, comparison function, decision model and measurement; the last one allows an estimation of the performance of the used decision model.

### 2.6.2 Open source tools

#### Febrl

Febrl[1] – Freely Extensible Biomedical Record Linkage – [Christen, 2008, 2009] is an open source tool written in Python and licensed under the Australian National University (ANU) Open Source License. Febrl was initially dedicated to data standardization and probabilistic record linkage in the biomedical domain. It includes a probabilistic approach based on a hidden Markov model for data standardization and seventh different methods for record linkage. Input data are read from files, and a graphical user interface is provided.

#### SQL Power DQguru

SQL Power DQguru[2] (formerly known as SQL Power MatchMaker) is a Java-based open source tool providing transformation and matching functions that users can use to define their own data matching criteria. The data conversion workflow can be manipulated by arranging and connecting individual process steps through a graphical user interface. Moreover, it performs duplicate verification and merging of duplicate records.

[1] http://sourceforge.net/projects/febrl
[2] http://www.sqlpower.ca/dqguru

SQL Power DQguru is distributed under the GNU General Public License (GPL) and can be interfaced with several database systems (e.g., Oracle, MySQL, PostgreSQL).

### Flamingo Package

The Flamingo Project, carried out at the University of California Irvine, focuses on how to deal with errors and inconsistencies in information systems and aims at developing algorithms in order to make query answering and information retrieval efficient in the presence of inconsistencies and errors.

The Flamingo package[3] [Li, 2011], developed in the context of the Flamingo Project, is an open source software supporting approximate string queries. It is implemented in C++ and includes algorithms for approximate selection queries, location-based approximate keyword search, selectivity estimation for approximate selection queries, and approximate queries on mixed types.

### Kettle

Kettle[4] (or Pentaho Data Integration Community Edition) is an open source tool written in Java for designing graphically ETL transformations and jobs, such as reading, manipulating, and writing data to and from various data sources. It is focusing primarily on connectivity and transformation, and it is easily extensible via Java plug-ins. Kettle supports a number of different database systems as well as a variety of flat files. It provides a set of predefined transformations, moreover users can implement further ones in JavaScript.

Kettle started as an independent open source ETL project and was later acquired by Pentaho to be included in the Pentaho Business Intelligence suite. It is released under the GNU Lesser General Public License (LGPL).

### Talend Open Studio

Talend Open Studio[5] is an open source data integration platform distributed under the GNU General Public License (GPL). It provides components for business process modeling and implements the ETL process in a graphical modeling environment. Through the graphical user interface users can drag and drop data processing components onto a process map. Once the design is completed, Talend Data Quality generates an executable code in Java or Perl that can be deployed and executed on

[3] http://flamingo.ics.uci.edu/index.html
[4] http://kettle.pentaho.com
[5] http://www.talend.com/products/talend-open-studio

data sources. Talend Open Studio contains a set of predefined transformations and users can specify additional ones in Java or Perl.

Talend Open Studio is developed by Talend, but it is also included in the open source Business Intelligence package from JasperSoft called JasperETL[6].

### CloverETL

CloverETL[7] is a Java-based data transformation and integration platform.

The engine of CloverETL is an open source Java library and does not provide the user interface component. It is released under the GNU Lesser General Public License (LGPL) and can be embedded in any application, commercial ones as well.

CloverETL is distributed in several editions: community, OEM program and commercials. The community edition (which is not open source) does not include the full set of transformation components and includes a limited version of the CloverETL Designer (i.e., the graphical user interface module).

### KETL

KETL[8], designed by Kinetic Networks, is an open source data integration platform with a Java-based architecture and an XML-based configuration. It supports extracting and loading of relational, flat file, and XML data sources, via JDBC and proprietary database APIs.

KETL and its features are released under a combination of the GNU Lesser General Public License (LGPL) and the GNU General Public License (GPL).

[6] http://community.jaspersoft.com/project/jaspersoft-etl
[7] http://www.cloveretl.com/products/open-source
[8] http://www.ketl.org

CHAPTER 3

# Data Quality Rules

As constraints and their enforcement play a key role in the maintenance of data integrity into a database, rules and their verification can play a key role in the evaluation and assessment of the consistency of a dataset: the consistency of a dataset can be defined in terms of constraints, and inconsistencies in the dataset itself can be detected as failures to comply with these constraints.

Classic integrity constraints are relevant for data quality and for data cleaning; however, they do not capture all the data quality concerns. For this reason, new forms of quality constraints, which can be considered an extension of usual semantic integrity constraints in databases, are proposed and investigated.

Moreover, data quality rules can play a further role: they can help in verifying the suitability of a dataset to be used for a certain purpose.

The rest of the chapter introduces different types of data constraints that can be used to evaluate the quality of a dataset and briefly mentions existing approaches to enforce data quality constraints.

## 3.1 Data Quality constraints

### 3.1.1 Data dependencies

Data dependencies, such as functional dependencies, are traditionally used for database schema design, integrity constraints, and query optimization, with respect to schema quality in databases [Elmasri and Navathe, 2000]. Recently, data dependencies have been revisited from the data quality perspective and used, for example, to capture data inconsistency, repair inconsistent data, and remove duplicates; moreover, new types of functional dependencies have been proposed as data quality constraints.

### Functional Dependencies

A Functional Dependency (FD) is a relationship between attributes of a database relation stating that the value of some attributes are uniquely determined by the values of some other attributes. More formally, given a relation schema $R = \{A_1, \ldots, A_n\}$ and a set of tuples $r$ from $dom(A_1) \times \cdots \times dom(A_n)$, where $dom(A)$ represents the domain of attribute $A$, a FD is a statement $X \rightarrow Y$ requiring that $X$ functionally determines $Y$, where $X, Y \subseteq R$. $X$ is called the left-hand side or the determinant of the dependency, and $Y$ is called the right-hand side or the dependent. The dependency is satisfied by a relation instance $r$ if, for all pair of tuples $t_1, t_2 \in r$, $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$ (where $t[X]$ denotes the projection of a tuple $t \in r$ to a subset $X \subseteq R$).

### Equality-Generating Dependencies

Equality-Generating Dependencies (EGDs), namely, dependencies over interpreted data, have been introduced in [Beeri and Vardi, 1984] as a generalization of FDs.

An EGD states that if some tuples fulfilling certain equalities exist in the database, some values in these tuples must be equal. Formally, an EGD is a pair $\langle (a_1, a_2), I \rangle$, where $a_1$ and $a_2$ are values for some attribute $A$ and $I$ is a finite relation such that $a_1, a_2 \in I[A]$. A relation $J$ satisfies $\langle (a_1, a_2), I \rangle$ if $h(a_1) = h(a_2)$ for any valuation $h$ such that $h(I) \subseteq J$; if $a_1 = a_2$, then it is trivially satisfied by every relation.

### Constraint-Generating Dependencies

In [Baudinet et al., 1999] a generalization of equality-generating dependencies, where equality requirements are replaced by arbitrary constraints on the data domain, have been introduced. Given a relation $r$, a constraint-generating $k$-dependency over $r$ (with $k \geq 1$) is a first-order formula of the form:

$$(\forall t_1) \cdots (\forall t_k)[[r(t_1) \wedge \cdots \wedge r(t_k) \wedge C[t_1, \ldots, t_k]] \Rightarrow C'[t_1, \ldots, t_k]]$$

where $C[t_1, \ldots, t_k]$ and $C'[t_1, \ldots, t_k]$ denote arbitrary constraint formulas relating the values of various attributes in the tuples $t_1, \ldots, t_k$. There are no restrictions on these formulas; they can include all constructs of the constraint theory under consideration, including constants and quantification on the constraint domain.

Constraint-generating 1-dependencies as well as constraint-generating 2-dependencies are the most common. For example, constraint-generating 1-dependencies can express a variety of arithmetic integrity constraints, while functional depen-

dencies and conditional functional dependencies can be expressed in the form of constraint-generating 2-dependencies.

### Conditional Functional Dependencies

In [Bohannon et al., 2007, Fan et al., 2008b] a class of constraints called Conditional Functional Dependencies (CFDs) has been proposed, and the application of this type of constraints in the data cleaning field has been studied. CFDs aim at capturing the consistency of data by incorporating bindings of semantically related values.

Given a relation schema $R$ defined over a fixed set of attributes denoted by $attr(R)$ with $dom(A)$ denoting the domain of each attribute $A \in attr(R)$, a CFD $\varphi$ on $R$ is defined as a pair $(R : X \rightarrow Y, T_p)$, where: (1) $X$ and $Y$ are sets of attributes from $attr(R)$; (2) $R : X \rightarrow Y$ is a standard FD, referred to as the FD embedded in $\varphi$; (3) $T_p$ is a tableau with attributes in $X$ and $Y$ (referred to as the pattern tableau of $\varphi$), where, for each $A$ in $X$ or $Y$ and each tuple t $\in T_p$, t[A] is either a constant '$a$' in $dom(A)$ or an unnamed variable denoted by '_' that draws values from $dom(A)$.

The pattern tableau $T_p$ of $\varphi$ refines the standard FD embedded in $\varphi$ by enforcing the binding of semantically related data values. $X$ can be denoted as $LHS(\varphi)$ and $Y$ as $RHS(\varphi)$; moreover, the attributes in a pattern tuple are separated with '||'.

In [Fan et al., 2008b] a *normal form* for CFDs has been defined: a CFD $\varphi$ is in normal form if $\varphi = (R : X \rightarrow A, t_p)$, where $A$ is a single attribute and the pattern tableau consists of a single pattern tuple $t_p$ only.

CFDs having only unnamed variables '_' in their tableaux are classic FDs, while CFDs with only constant values in their tableaux are called *constant* CFDs and can be considered equivalent to association rules.

### Constrained Functional Dependencies

Constrained functional dependencies have been proposed by [Maher, 1997]. They extend the traditional notion of FDs by expressing that a FD holds on a subset of a relation, where the subset is defined by a constraint.

These dependencies have been proposed to be used with logic programming languages and to express constraints in databases. Although they are based on the same idea on which CFDs are founded, they were not used in the data cleaning field.

### CFD extensions

After the publication of the works about CFDs, extensions of CFDs have been proposed to support disjunction and negation [Bravo et al., 2008], to specify patterns

in terms of value ranges [Golab et al., 2008], and to specify patterns of data values containing built-in predicates [Chen et al., 2009].

***Extended Conditional Functional Dependencies***

The Extended Conditional Functional Dependency (eCFD) has been proposed in [Bravo et al., 2008]. In contrast to CFDs, eCFDs specify patterns of semantically related values in terms of disjunction and inequality, and they can catch inconsistencies that cannot be detected by CFDs.

Given a relation schema $R$ defined over a fixed set of attributes denoted by $attr(R)$ with $dom(A)$ denoting the domain of each attribute $A \in attr(R)$, an eCFD $\varphi$ is a triple $(R : X \rightarrow Y,\ Y_p,\ T_p)$, where (1) $X, Y, Y_p \subseteq attr(R)$, and $Y \cap Y_p = \emptyset$; (2) $X \rightarrow Y$ is a standard FD, referred to as the embedded FD of $\varphi$; (3) $T_p$ is a pattern tableau consisting of a finite number of pattern tuples over the attributes in $X \cup Y \cup Y_p$ such that, for any tuple $t_p \in T_p$ and for each attribute $A$ in $X \cup Y \cup Y_p$, $t_p[A]$ is either an unnamed variable '_', a set $S$, or a complement set $\bar{S}$, where $S$ is a finite subset of $dom(A)$. $X$ is denoted by $LHS(\varphi)$ and $Y \cup Y_p$ by $RHS(\varphi)$.

***Range Tableaux***

In [Golab et al., 2008] a range tableau $T_r$ is defined as a tableau with all attributes from $X$ and $Y$, where, for each row $t_r \in T_r$ and each (ordered) attribute $A \in X \cup Y$, $t_p[A] = [a_1, a_2]$, with $a_1, a_2 \in dom(A)$ and $a_1 \leq a_2$. Ranges generalize both constants and wildcards, as a constant '$a$' can be written $[a, a]$ and '_' can be written $[a_{min}, a_{max}]$, where $a_{min} = \min\ dom(A)$ and $a_{max} = \max\ dom(A)$.

To denote that a tuple $t \in dom(R)$ satisfies a particular row $t_p$ of tableau $T$, the symbol $\asymp$ is used. Given a tuple $t \in dom(R)$ and a row $t_r \in T_r$, $t[S] \asymp t_r[S]$ iff, for each attribute $A$ of $S$, $t[A] \in t_r[A]$; namely, $t$ matches the tableau row $t_r$ if $t[A]$ is an element of the range $t_r[A]$ for every attribute $A$.

A relation instance $dom(R)$ satisfies a CFD $\varphi = (R : X \rightarrow Y,\ T_r)$ iff $\forall t_i, t_j \in dom(R)$ and $\forall t_r \in T_r$ if $t_i[X] = t_j[X] \asymp t_r[X]$, then $t_i[Y] = t_j[Y] \asymp t_r[Y]$.

***CFD$^{\mathrm{p}}$s***

A type of data dependency called CFD$^{\mathrm{p}}$ has been presented in [Chen et al., 2009]. CFD$^{\mathrm{p}}$s are CFD with built-in predicates ($\neq$, $<$, $>$, $\leq$, $\geq$) in the patterns of data values. A CFD$^{\mathrm{p}}$ $\varphi$ on R is a pair $R(X \rightarrow Y,\ T_p)$, where: (1) $X$ and $Y$ are sets of attributes in $attr(R)$; (2) $X \rightarrow Y$ is a standard FD, referred to as the FD embedded in $\varphi$; (3) $T_p$ is a tableau with attributes in $X$ and $Y$, referred to as the pattern tableau of $\varphi$, where, for each $A$ in $X \cup Y$ and each tuple $t_{p_i} \in T_p$, $t_{p_i}[A]$ is either an unnamed variable '_' that draws values from $dom(A)$ or '$op\ a$', where $op$ is one of $\neq$, $<$, $>$, $\leq$, $\geq$, and '$a$' is a constant in $dom(A)$.

Intuitively, each pattern tuple $t_{p_i}$ specifies a condition via $t_{p_i}[X]$, and the dependency is satisfied if $t[X]$ satisfies the conjunction of all these conditions. CFD$^{\text{P}}$s are more powerful than CFDs but cannot express disjunctions as defined in [Bravo et al., 2008].

### Differential Dependencies

In a recent paper [Song and Chen, 2011], a new kind of dependency, called differential dependency, has been proposed. Differential dependencies specify constraints on the differences of the values of the attributes and can be used as a novel class of integrity constraints.

Formally, given a relation $r$ over a schema $R$, a differential dependency written in the form $\phi_L[X] \rightarrow \phi_R[Y]$, where $\phi_L[X]$ and $\phi_R[Y]$ are differential functions specifying constraints on distances over attribute sets $X$ and $Y$ of $R$. It states that, for any two tuples from $r$, if their value differences – measured by certain distance metric – on attributes in $X$ agree the differential function $\phi_L[X]$ (i.e., distance constraints on $X$), then their value differences on attributes in $Y$ should also agree with the differential function $\phi_R[Y]$.

### Order Dependencies

Order dependencies can capture a monotonicity property between two sets of values projected onto some attributes in a relation: given a relation schema $R(U)$, an order dependency $X \rightarrow Y$ (with $X, Y \subseteq U$) holds if an order over the values of each attribute in $X$ implies an order over the values of each attributes of $Y$.

Order dependencies were introduced for the first time in the context of database systems by [Ginsburg and Hull, 1983]. Later, *Ordered Functional Dependencies* (OFDs) have been defined in the context of an extension of the relational data model to incorporate ordered domains [Ng, 1999].

The semantics of OFDs has been defined by means of two orderings, pointwise ordering and lexicographical ordering, and the corresponding OFDs have been called respectively *Pointwise Ordered Functional Dependencies* (POFDs) and *Lexicographical Ordered Functional Dependencies* (LOFDs). According to a pointwise ordering, the tuple $x_1, \ldots, x_n$ is less than another tuple $y_1, \ldots, y_n$ if $x_i \leq y_i$ for each $1 \leq i \leq n$; while, according to a lexicographical ordering, the tuple $x_1, \ldots, x_n$ is less than another tuple $y_1, \ldots, y_n$ if there is an index $j \geq i$ such that $x_j < y_j$ and $x_i = y_i$ for each $i < j$.

Even if order dependencies have been proposed in the database context, we are not aware of any application of these dependencies in the data cleaning field. In a

recent work, however, order dependencies have been used in the query optimization context [Szlichta et al., 2012].

### 3.1.2 Existence constraints

In a dataset the presence or absence of values for some attributes may be related to the presence or absence of values for other attributes. In order to deal with this kind of relations, two types of constraints, called existence constraints and disjunctive existence constraints, were introduced in literature works related to databases with incomplete information [Atzeni and Morfuni, 1986].

Considering that a tuple is said to be total on $X$ (or $X$-total) if each attribute $A \in X$ is not null, an existence constraint is a statement $e : X \rightarrow Y$ (read $X$ requires $Y$), where $X$ and $Y$ are sets of attributes, holding in a relation $r$ over a scheme $R(U)$ (where $U$ is a finite set of attributes and $X \cup Y \subseteq U$) if each $X$-total tuple $t \in r$ is also $Y$-total. If $Y = \{\ \}$, then the existence constraint is assumed satisfied.

A disjunctive existence constraint is a statement $d : X \rightarrow S$, where $X$ is a set of attributes and $S = Y_1, Y_2, \ldots, Y_n$ is a set of sets of attributes; $X \rightarrow Y_1, Y_2, \ldots, Y_n$ holds in a relation $r$ over a scheme $R(U)$ (with $X, Y_1, Y_2, \ldots, Y_n \subseteq U$) if, for each $X$-total tuple $t \in r$, there is an $i \in 1, 2, \ldots, n$ such that $t$ is $Y_i$-total.

In the relational database design context, the term existence dependency has been used as a synonym of participation constraint [Elmasri and Navathe, 2000], a structural constraint indicating that the existence of an entity depends on its being related to another entity. In a relational database, a participation constraint can be implemented as a foreign key integrity constraint.

### 3.1.3 Association rules

Some literature works propose to use association rules extracted from a dataset as a means to discover dirty data; for example, in [Hipp et al., 2001] a rule-based data mining approach for outlier detection is described.

An association rule has the form $X \Rightarrow Y$, where $X$ and $Y$ are two sets of items; $X$ is called left-hand side and $Y$ right-hand side.

The idea to look for association rules in a dataset was originally introduced in the "basket" data analysis field, where purchased data were grouped by the purchase transactions, and associations between two sets of bought items were found. In general, the data mining process consists in discovering associations between two sets of data found in the same group. As the number of associations might be huge, and not all the discovered associations are meaningful, two probability measures – called support and confidence – are introduced to discard the less frequent associations in a

repository. The support is the joint probability to find $X$ and $Y$ in the same group; the confidence is the conditional probability to find in a group $Y$ having found $X$. Two thresholds, respectively for support and confidence, are given in input to the data mining process to discard the less frequent association rules.

## 3.2 Data constraint enforcement

Basic integrity constraints can be easily enforced in a relational database using the features available in the SQL language. For example, domain constraints and constraints based on foreign keys can be specified in the creation of relational tables.

In order to express more complex constraints, the specifications of the standard SQL-92 include a predicate, called *assertion*, for the definition of schema-level constraints [Elmasri and Navathe, 2000]. The *assertion* predicate can express a condition over the schema that has to be always satisfied. However, the existing database management systems do not implement the *assertion* predicate; this is probably due to the performance problems inherent with the implementation of a predicate that can be arbitrarily complex and that has to be executed at every change occurring in the database.

### 3.2.1 Active databases

Using a different approach, complex constraints can be implemented in the *active databases* through the functionalities they provide for the definition of active rules.

In active databases, it is possible to enforce several constraints by specifying the type of event that may cause the constraints to be violated and evaluating appropriate conditions that check if the constraints are violated by the event or not. Also complex application constraints, sometimes called business rules, can be enforced with this approach.

The common way in which active rules are implemented in active databases is through triggers, which are statements automatically executed by the database management system whenever a specified event occurs. Triggers are offered by almost all the commercial database management systems and are commonly used for data integrity maintenance.

### 3.2.2 Business rule management systems

A business rule management system [Bajec et al., 2000] is a software system used to define, execute, and monitor the decision logic that is used by operational systems within an organization. The decision logic is represented through business rules

that are meant to capture knowledge of constraints in an organization. They reflect the business policy of an enterprise; however, while a business policy is a general statement or direction for an organization, a business rule defines or constrains some aspects of a business, asserts business structures, and influences the behavior of an enterprise.

This kind of systems can contribute in improving the quality of the data produced by the business processes, even if data quality is not their main target.

Three basic types of business rules have been identified in literature [Taveter and Wagner, 2001]: integrity rules (also called integrity constraints), derivation rules (also called deduction rules or Horn clauses), and reaction rules (also called action rules or event-condition-action rules). A fourth type, called deontic assignments, has only been marginally discussed in a proposal of considering authorizations as business rules.

From an information system perspective, integrity rules (the first cited type of business rules) play the role of constraints on a database helping ensure that the structure and content of the real world is incorporated into the database.

The classic approach to computational systems based on rules comes from works on logic programming and deductive databases. The current commercial business rule systems seek to apply aspects of rule technology to implement rule-based logic into specific business processes.

In some systems, business rules are implemented with the SQL language and are stored as part of the database definition, but most of the rule engines use their own proprietary rule specification language. The main differences among these languages lie in the types of rules they support and in the way rules are recorded.

The approach based on business rules is widely adopted by software vendors, who provide business process modeling tools with the ability to explicitly define the constraints on the processes. There are also open source initiatives, mostly based on Java technologies, such as Jess Rule Engine, JBoss Drools, JRuleEngine, Hammurapi Rules, and SweetRules.

CHAPTER 4

# Evaluation of Data Quality Rules

In order to use data quality rules to evaluate the quality of a dataset, we addressed two main targets: the expression of quality concerns and their evaluation against concrete instances.

The solution applied to the case study consists, as a first step, in the enforcement of basic constraints during the insertion of the data into the database and, as a second step, when the whole dataset is available, in the verification of more complex data quality rules through the software tool we developed.

The aim of the tool is to identify the tuples not satisfying the given rules using a simple approach in the definition of the rules themselves and without requiring to the final users any software development.

More formally, the error detection problem can be expressed in the following way: considering a database schema $R = (R_1, \ldots, R_n)$, where $R_i$ is a relation schema with $i \in [1, n]$, and having in input a set $S$ of data quality rules defined on $R$ and a database instance $D = (I_1, \ldots, I_n)$ of $R$, the error detection problem consists in finding the subset $(I'_1, \ldots, I'_n)$ of $D$ such that $\forall i \in [1, n]$ $I'_i \subseteq I_i$ and each tuple in $I'_i$ violates at least one data quality rule in $S$.

The developed tool, which is Java technology based, provides functionalities to express data quality rules and to evaluate them against a dataset stored in a relational database identifying the subsets of data that do not comply with the defined rules.

In the tool, rules are expressed using the XML markup language [Harold and Means, 2004]. The root of the XML document is called `rules`; under the root, each rule is specified through a tag called `rule_definition`. The Document Type Definition (DTD) and the XML Schema for the XML structure used to express the rules are shown in appendix A.

In order to facilitate the input of the rules, predefined templates are available

for each type of rule accepted by the tool; after a template has been completed with the details of a rule, the correct XML code for that rule is automatically generated by the tool.

The data to be checked are stored in a relational database; in particular, for the case study we used PostgreSQL[1]. In the rules, it is necessary to specify the tables containing the data; instead of tables, views can be used as well, allowing more flexibility in the definition of data constraints. For each defined rule, the tool identifies all the records that do not comply with the rule itself; then the identified records are stored in views or tables inside the same database containing the dataset.

## 4.1 Types of rules

The developed tool manages rules of the following types:

**Conditional Constraint rule** To express constraints that can be written in the form `if-then` among attributes of the same relation. Association rules, constant CFDs, and constant CFD$^{\text{p}}$s can be expressed with this type of rule.

**Functional Dependency rule** To define FDs among attributes of the same relation. Non-constant CFDs and non-constant CFD$^{\text{p}}$s can also be specified with this kind of rule.

**Order Dependency rule** To express order constraints among attributes belonging to the same relation.

**Differential Dependency rule** To specify differential dependencies among numerical attributes of the same relation.

**Existence Constraint rule** To define existence constraints among attributes belonging to the same or to different relations.

**Check Constraint rule** It consists of boolean conditions, which can contain arithmetic operators, on single attributes or among attributes belonging to the same or to different relations.

In the following, we detail the rule types managed by the tool showing the corresponding XML syntax and providing examples of SQL statements that can be used to identify the records that do not comply with the rules. In the SQL statements shown in the following, XML tags are referred using XPath-like expressions and considering only the piece of XML code used in each example.

[1] http://www.postgresql.org

### 4.1.1 Conditional Constraint rules

This type of rule can be used to express a constraint that has to be valid if a given condition is satisfied; namely, a constraint that can be written in the form `if-then`. For example, association rules, constant CFDs, and constant CFD$^p$s (i.e., CFDs and CFD$^p$s with constant patterns only [Fan et al., 2011]) can be expressed with this kind of rule.

The following XML skeleton is used to express this type of rule:

```
<rule_definition type="conditional_rule">
  <table_name></table_name>
  <rule_cr>
    <if></if>
    <then></then>
  </rule_cr>
</rule_definition>
```

The `if` and `then` components of the rule contain one or more conditions, which can contain arithmetic operators, connected by means of boolean operators.

The XML structure used to express the conditions is detailed in the following DTD fragment:

```
<!ELEMENT then (condition|conditions)>
<!ELEMENT when (condition|conditions)>
<!ELEMENT conditions ((condition|conditions)+, boolean_operator)>
<!ELEMENT condition (lside, comparison_operator, rside)>
<!ELEMENT lside (column|arithmetic_computation)>
<!ELEMENT rside (constant|column|arithmetic_computation)>
<!ELEMENT column (table_name?, column_name)>
<!ELEMENT arithmetic_computation ((constant|column|
  arithmetic_computation)+, arithmetic_operator)>
```

The tag `comparison_operator` can contain one of the following comparison operators: $=$, $\neq$, $<$, $>$, $\leq$, $\geq$; the tag `boolean_operator` can contain the boolean operators AND and OR; finally, the tag `arithmetic_operator` can contain one of the following arithmetic operators: +, -, *, /.

In the following, we provide an example of SQL statement that can be used to find the records that do not comply with a conditional constraint rule:

```
SELECT /rule_definition/rule_cr//column_name
FROM /rule_definition/table_name
```

```
WHERE /rule_definition/rule_cr/if
AND NOT /rule_definition/rule_cr/then
```

### 4.1.2 Functional Dependency rules

Functional dependencies among attributes of the same relation can be defined with this type of rule, including CFDs and CFD$^p$s expressed in their *normal form* as defined in [Fan et al., 2008b] (for the definition of *normal form* for CFDs refer to chapter 3).

The XML skeleton to specify this kind of rule is the following:

```
<rule_definition type="functional_dependency">
  <table_name></table_name>
  <rule_fd>
    <lhs>
      <column_name></column_name> +
    </lhs>
    <rhs>
      <column_name></column_name> +
    </rhs>
    <when></when> ?
  </rule_fd>
</rule_definition>
```

The left-hand side and the right-hand side of the rule can contain one or more attributes. In order to express CFDs and CFD$^p$s, it is possible to add conditions connected by means of boolean operators; the conditions have the same XML structure used for the conditions in the conditional constraint rules, and they are collected under the tag `when`.

The following SQL statement[2] exemplifies the way in which the tuples that do not satisfy a functional dependency rule can be identified:

```
WITH tmpView AS (
  SELECT //lhs/column_name, COUNT (DISTINCT (//rhs/column_name))
  FROM /rule_definition/table_name
  WHERE //when
  GROUP BY //lhs/column_name
  HAVING COUNT (DISTINCT (//rhs/column_name))>1
```

[2]The SQL *WITH* clause implements the Common Table Expression (CTE) defined in the SQL:1999 standard.

```
)
SELECT DISTINCT tableName.//lhs/column_name, //rhs/column_name
FROM tmpView, //rule_definition/table_name As tableName
WHERE tmpView.//lhs/column_name = tableName.//lhs/column_name
AND //when
```

### 4.1.3 Order Dependency rules

Adopting the definition of order dependency provided in literature [Ginsburg and Hull, 1983], we defined a rule type to express direct and inverse order constraints among attributes belonging to the same relation.

More formally, given a relation schema $R(U)$ and a relation instance $r$ over $R$, this type of rule can be used to define a dependency $X \rightarrow Y$ $(X, Y \subseteq U)$ such that:

- $X \rightarrow_{\preceq} Y$ denotes a *direct* order dependency if, for every pair of tuples $s$ and $t \in r$, $s[X] \preceq t[X]$ implies $s[Y] \preceq t[Y]$, where $s[X] \preceq t[X]$ if $s[A] \leq t[A]$ $\forall A \in X$ and $s[Y] \preceq t[Y]$ if $s[B] \leq t[B]$ $\forall B \in Y$.
- $X \rightarrow_{\succeq} Y$ denotes an *inverse* order dependency if, for every pair of tuples $s$ and $t \in r$, $s[X] \preceq t[X]$ implies $s[Y] \succeq t[Y]$, where $s[X] \preceq t[X]$ if $s[A] \leq t[A]$ $\forall A \in X$ and $s[Y] \succeq t[Y]$ if $s[B] \geq t[B]$ $\forall B \in Y$.

Furthermore, we introduced the possibility to apply these dependencies on non-overlapping subsets of tuples; given $P \subset R$:

- $(X \rightarrow_{\preceq_P} Y|P)$ denotes a *direct* order dependency on a partition of $r$ if, for every pair of tuples $s$ and $t \in r$, $s[P] = t[P]$ and $s[X] \preceq t[X]$ imply $s[Y] \preceq t[Y]$, where $s[Y] \preceq t[Y]$ if $s[A] \leq t[A]$ $\forall A \in Y$.
- $(X \rightarrow_{\succeq_P} Y|P)$ denotes an *inverse* order dependency on a partition of $r$ if, for every pair of tuples $s$ and $t \in r$, $s[P] = t[P]$ and $s[X] \preceq t[X]$ imply $s[Y] \succeq t[Y]$, where $s[Y] \succeq t[Y]$ if $s[A] \geq t[A]$ $\forall A \in Y$.

The following XML skeleton expresses direct and inverse order dependencies:

```
<rule_definition type="order_dependency">
  <table_name></table_name>
  <rule_od type="direct|inverse">
    <lhs>
      <column_name></column_name> +
    </lhs>
    <rhs>
```

```
      <column_name></column_name> +
    </rhs>
    <when></when> ?
    <partition_on></partition_on> ?
  </rule_od>
</rule_definition>
```

The scope of the rule can be limited to a subset of the relation instance through conditions, which can be collected in the XML format under the tag `when`. In addition, in order to express order dependencies on non-overlapping sets of tuples, it is possible to specify these sets on the basis of attributes, which in XML are expressed using the tag `partition_on`.

The following SQL statement identifies the records that do not satisfy a *direct* order constraint rule and provides, for each selected record, the number of failed comparisons with the other records:

```
WITH tmpView AS (
  SELECT DISTINCT //lhs/column_name, //rhs/column_name,
         //rule_od/partition_on/column_name AS partCol
  FROM /rule_definition/table_name
  WHERE //rule_od/when
)
SELECT tableName.//lhs/column_name,
       tableName.//rhs/column_name, COUNT(*)
FROM tmpView, /rule_definition/table_name AS tableName
WHERE //rule_od/partition_on/column_name[j] =
      tmpView.partCol[j] AND //rule_od/when
AND ((tableName./lhs/column_name[1] <=
      tmpView.//lhs/column_name[1]
AND   tableName./lhs/column_name[i] <=
      tmpView.//lhs/column_name[i]
AND (NOT tableName.//rhs/column_name[1] <=
         tmpView.//rhs/column_name[1]
OR NOT tableName.//rhs/column_name[k] <=
       tmpView.//rhs/column_name[k]))
OR (tmpView.//lhs/column_name[1] <=
    tableName./lhs/column_name[1]
AND tmpView.//lhs/column_name[i] <=
    tableName./lhs/column_name[i]
```

```
AND (NOT tmpView.//rhs/column_name[1] <=
         tableName.//rhs/column_name[1]
OR NOT tmpView.//rhs/column_name[k] <=
       tableName.//rhs/column_name[k])))
GROUP BY tableName.//lhs/column_name, tableName.//rhs/column_name
```

Furthermore, the following SQL statement can be used to find the records that do not satisfy an *inverse* order constraint rule with, for each selected record, the number of failed comparisons with the other records:

```
WITH tmpView AS (
  SELECT DISTINCT //lhs/column_name, //rhs/column_name,
         //rule_od/partition_on/column_name AS partCol
  FROM /rule_definition/table_name
  WHERE //rule_od/when
)
SELECT tableName.//lhs/column_name,
       tableName.//rhs/column_name, COUNT(*)
FROM tmpView, /rule_definition/table_name AS tableName
WHERE //rule_od/partition_on/column_name[j] =
      tmpView.partCol[j] AND //rule_od/when
AND ((tableName./lhs/column_name[1] <=
      tmpView.//lhs/column_name[1]
AND   tableName./lhs/column_name[i] <=
      tmpView.//lhs/column_name[i]
AND (NOT tableName.//rhs/column_name[1] >=
         tmpView.//rhs/column_name[1]
OR NOT tableName.//rhs/column_name[k] >=
       tmpView.//rhs/column_name[k]))
OR (tmpView.//lhs/column_name[1] <=
    tableName./lhs/column_name[1]
AND tmpView.//lhs/column_name[i] <=
    tableName./lhs/column_name[i]
AND (NOT tmpView.//rhs/column_name[1] >=
         tableName.//rhs/column_name[1]
OR NOT tmpView.//rhs/column_name[k] >=
       tableName.//rhs/column_name[k])))
GROUP BY tableName.//lhs/column_name, tableName.//rhs/column_name
```

### 4.1.4 Differential Dependency rules

This rule specifies differential dependencies [Song and Chen, 2011]; namely, dependencies that can be used to specify constraints on distances over attributes of the same relation.

In the developed tool, we limited the application of differential dependencies only to numerical attributes, and we introduced the possibility to apply these dependencies on non-overlapping sets of tuples.

The XML skeleton to express differential dependencies is the following:

```
<rule_definition type="distance_dependency">
  <table_name></table_name>
  <rule_dd>
    <lhs_dd>
      <distance_condition> +
        <column_name></column_name>
        <comparison_operator></comparison_operator>
        <constant></constant>
      </distance_condition>
    </lhs_dd>
    <rhs_dd>
      <distance_condition> +
        <column_name></column_name>
        <comparison_operator></comparison_operator>
        <constant></constant>
    </rhs_dd>
    <when></when> ?
    <partition_on></partition_on> ?
  </rule_dd>
</rule_definition>
```

The tag `when` can contain conditions and can be used to reduce the scope of the rule to a subset of the relation instance. In addition, with the tag `partition_on` it is possible to specify differential dependencies for non-overlapping sets of tuples on the basis of the attributes specified in this tag.

The following SQL statement identifies the tuples that do not satisfy a differential dependency rule and provides, for each selected tuple, the number of failed comparisons with the other tuples:

```
WITH tmpView AS (
```

```
  SELECT //lhs_dd//column_name, //rhs_dd//column_name
         //partition_on/column_name AS partCol
  FROM /rule_definition/table_name
  AND //when
)
SELECT tableName.//lhs_dd//column_name,
       tableName.//rhs_dd//column_name, COUNT(*)
FROM tmpView, /rule_definition/table_name AS tableName
WHERE //partition_on/column_name[j] = tmpView.partCol[j]
AND //when
AND
  (abs(tableName.//lhs_dd/distance_condition[i]/column_name -
  tmpView.//lhs_dd/distance_condition[i]/column_name)
  //lhs_dd/distance_condition[i]/comparison_operator
  //lhs_dd/distance_condition[i]/constant
AND (NOT
  abs(tableName.//rhs_dd/distance_condition[1]/column_name -
  tmpView.//rhs_dd/distance_condition[1]/column_name)
  //rhs_dd/distance_condition[1]/comparison_operator
  //rhs_dd/distance_condition[1]/constant
OR NOT
  abs(tableName.//rhs_dd/distance_condition[k]/column_name -
  tmpView.//rhs_dd/distance_condition[k]/column_name)
  //rhs_dd/distance_condition[k]/comparison_operator
  //rhs_dd/distance_condition[k]/constant))
GROUP BY tableName.//lhs/column_name, tableName.//rhs/column_name
```

### 4.1.5 Existence Constraint rules

Referring to the definition of existence constraints introduced in works related to databases with incomplete information [Atzeni and Morfuni, 1986], we defined existence constraint rules among attributes belonging to the same or to different relations.

Two types of existence constraints, called existence constraints and disjunctive existence constraints, have been defined in literature [Atzeni and Morfuni, 1986]. Unlike the definitions proposed in the cited work, the rule implemented in the tool manages only attributes instead of sets of attributes in the left-hand side of disjunctive existence constraints and in both sides of existence constraints.

More formally, given a relation schema $R(U)$ and a relation instance $r$ over $R$, this type of rule can express dependency-like existence constraints and disjunctive-like existence constraints defined as:

- A dependency-like existence constraint $A \rightarrow B$ (read $A$ requires $B$), where $A, B \in U$, holds on $r$ if, for each tuple $t \in r$, $t[A] \neq$ null implies $t[B] \neq null$.
- A disjunctive-like existence constraint $A \rightarrow S$, where $S = \{Y_1, \ldots, Y_n\}$ is a set of sets of attributes (with $Y_1, \ldots, Y_n \subseteq U$) and $A \in U$, holds on $r$ if, for each tuple $t \in r$, $t[A] \neq null$ implies $\exists\ Y \in S$ such that $\forall B \in Y\ t[B] \neq null$.

In addition, this rule type allows the definition of existence constraints on single attributes and bidirectional existence constraints between two attributes (i.e., if one of the two attributes exists in the dataset, the second has to exist as well) contained in the same table or in different tables.

The XML skeletons to express the four types of existence constraints managed by the tool are shown in the following:

```
<rule_definition type="existence_constraint">
  <table_name></table_name> +
  <rule_ec type="ec_dep">
    <lhs_ec>
      <column></column>
    </lhs_ec>
    <rhs_ec>
      <column></column>
    </rhs_ec>
    <when></when> ?
    <join_on></join_on> ?
  </rule_ec>
</rule_definition>

<rule_definition type="existence_constraint">
  <table_name></table_name> +
  <rule_ec type="ec_disj">
    <lhs_ec>
      <column></column>
    </lhs_ec>
    <rhs_ec>
      <table_name></table_name> ?
```

```
      <disj_attr> +
        <column_name></column_name> +
      </disj_attr>
    </rhs_ec>
    <when></when> ?
    <join_on></join_on> ?
  </rule_ec>
</rule_definition>

<rule_definition type="existence_constraint">
  <table_name></table_name> +
  <rule_ec type="ec_bidir">
    <column></column> +
    <when></when> ?
    <join_on></join_on> ?
  </rule_ec>
</rule_definition>

<rule_definition type="existence_constraint">
  <table_name></table_name>
  <rule_ec type="ec_attr">
    <column></column>
    <when></when>
  </rule_ec>
</rule_definition>
```

The scope of the rule can be limited to a subset of the relation instance through conditions, which can be expressed in the XML format using the tag `when`. In addition, when two tables are referred in the rule, it is required to specify the criteria to be used to join them; in the XML format, joining criteria are labeled with the tag `join_on`.

The following SQL statements can be used to identify the records that do not comply with dependency-like, disjunctive-like, and bidirectional existence constraints when only one table is referred in the rule:

```
SELECT //rule_ec//column_name
FROM //rule_definition/table_name
WHERE //lhs_ec//column_name IS NOT NULL
AND //rhs_ec//column_name IS NULL
AND //rule_ec/when
```

```
SELECT //rule_ec//column_name
FROM //rule_definition/table_name
WHERE //lhs_ec//column_name IS NOT NULL
AND (//rhs_ec/disj_attr[1]/column_name[1] IS NULL
OR //rhs_ec/disj_attr[1]/column_name[k] IS NULL)
AND (//rhs_ec/disj_attr[i]/column_name[1] IS NULL
OR //rhs_ec/disj_attr[i]/column_name[k] IS NULL)
AND //rule_ec/when
```

```
SELECT //rule_ec//column_name
FROM //rule_definition/table_name
WHERE (//rule_ec/column[1]/column_name IS NULL
AND //rule_ec/column[2]/column_name IS NOT NULL)
OR (//rule_ec/column[2]/column_name IS NULL
AND //rule_ec/column[1]/column_name IS NOT NULL)
AND //rule_ec/when
```

Furthermore, SQL statements that can be used to find the records that do not comply with dependency-like, disjunctive-like, and bidirectional existence constraints when the attributes used in the rule belong to different tables, are listed in the following:

```
SELECT //rule_ec//column_name
FROM //lhs_ec//table_name
LEFT OUTER JOIN //rhs_ec//table_name
ON (//rule_ec/join_on)
WHERE //lhs_ec//column_name IS NOT NULL
AND //rhs_ec//column_name IS NULL
AND //rule_ec/when
```

```
SELECT //rule_ec//column_name
FROM //lhs_ec//table_name
LEFT OUTER JOIN //rhs_ec/table_name
ON (//rule_ec/join_on)
WHERE //lhs_ec//column_name IS NOT NULL
AND (//rhs_ec/disj_attr[1]/column_name[1] IS NULL
OR //rhs_ec/disj_attr[1]/column_name[k] IS NULL)
AND (//rhs_ec/disj_attr[i]/column_name[1] IS NULL
OR //rhs_ec/disj_attr[i]/column_name[k] IS NULL)
AND //rule_ec/when
```

```
SELECT //rule_ec//column_name
FROM //rule_ec/column[1]/table_name
FULL OUTER JOIN //rule_ec/column[2]/table_name
ON (//rule_ec/join_on)
WHERE (//rule_ec/column[1]/column_name IS NULL
AND //rule_ec/column[2]/column_name IS NOT NULL)
OR (//rule_ec/column[2]/column_name IS NULL
AND //rule_ec/column[1]/column_name IS NOT NULL)
AND //rule_ec/when
```

Finally, an SQL statement to select the records that do not comply with existence constraints defined on single attributes is:

```
SELECT //rule_ec//column_name
FROM //rule_definition/table_name
WHERE //rule_ec//column_name NOT NULL
AND //rule_ec/when
```

### 4.1.6 Check Constraint rules

A check constraint rule consists of boolean conditions, which can contain arithmetic operators, on single attributes or among attributes belonging to the same or to different relations.

The following XML skeleton is used to express this type of rule:

```
<rule_definition type="check_constraint">
  <table_name></table_name> +
  <rule_cc>
    <check></check>
    <join_on></join_on> ?
  </rule_cc>
</rule_definition>
```

One or more conditions, connected by means of boolean operators, can be defined under the tag `check`. Moreover, when more than one table is used, it is required to specify the criteria to be applied to join the tables themselves; in the XML format, joining criteria are labeled with the tag `join_on`.

An example of SQL statement that can be used to identify the records that do not comply with the rule when the referred attributes belong to the same table, is the following:

```
SELECT /rule_definition/rule_cc/column_name
FROM /rule_definition/table_name
WHERE NOT /rule_definition/rule_cc/check
```

By contrast, the following SQL statement can be used when more than one table is considered in the rule:

```
SELECT /rule_definition/rule_cc//column_name
FROM /rule_definition/join/column[1]/table_name
JOIN /rule_definition/join/column[2]/table_name
ON /rule_definition/join/column[1]/column_name =
   /rule_definition/join/column[2]/column_name =
JOIN /rule_definition/join/column[k]/table_name
ON /rule_definition/join/column[1]/column_name =
   /rule_definition/join/column[k]/column_name =
WHERE NOT /rule_definition/rule_cc/check
```

## 4.2 XML validation with Schematron

The validation of an XML document through XML Schema ensures that the XML code conforms to the given schema in terms of elements and structure. It is possible to define exactly which elements and attributes can appear in an XML document, in which order they will appear, and which types of data they can contain. This makes XML Schema a grammar-based schema language because it deals with the structure of things, just as grammar deals with the structure of things in spoken languages.

Often, however, structure is not enough, and there is the need to introduce and check other types of constraints. For this purpose, in addition to XML Schema, we used Schematron [van der Vlist, 2007], which is a structural based validation language. Schematron uses a tree pattern based paradigm, rather than the regular grammar paradigm used in DTD and XML Schema. Tree patterns, defined as XPath expressions, are used to make assertions and provide reports about XML documents. Through the exploitation of this paradigm, Schematron provides the power to express constraints across elements and attributes and also among their values.

The Schematron reference implementation is actually an XSLT transformation which transforms a Schematron schema into an XSLT document that is then used to validate an XML document. Several implemented versions of Schematron are currently available; namely, Schematron 1.5, Schematron 1.6 and ISO Schematron

(Schematron has been standardized by ISO/IEC to become part of ISO/IEC 19757 DSDL). In our work, we used the ISO version.

In the following, we list some of the main constraints defined to validate the XML structure we used to specify the data quality rules, with the Schematron code to express them:

– In order to check the correct association of the tag corresponding to the chosen rule type (i.e., `rule_cr`, `rule_fd`, `rule_od`, `rule_dd`, `rule_ec`, `rule_cc`) with the attribute `type` in the `rule_definition` tag (we report only the assertion for the conditional constraint rule because the others have the same structure):

```
<iso:rule context="rule_definition">
  <iso:assert test="(@type='conditional_rule' and rule_cr)
   or @type!='conditional_rule'">
    For a conditional rule the tag rule_cr is required
  </iso:assert>
</iso:rule>
```

– In rules of type *conditional rule*, *functional dependency*, *order dependency*, and *differential dependency* only one table name has to be provided:

```
<iso:rule context="rule_definition">
  <iso:assert test="((@type='conditional_rule' or
   @type='functional_dependency' or @type='order_dependency' or
   @type='distance_dependency') and count(table_name) = 1) or
   (@type!='conditional_rule'and @type!='functional_dependency'
   and @type!='order_dependency'
   and @type!='distance_dependency')"> For conditional rules,
    functional dependencies, order dependencies and differential
    dependencies only one table name is allowed
  </iso:assert>
</iso:rule>
```

– In an *existence constraint* rule, it is possible to define one or two table names; moreover, when two table names are present, they have to be distinct:

```
<iso:rule context="rule_definition">
  <iso:assert test="(@type='existence_constraint'
   and (count(table_name)=1 or count(table_name)=2))
   or @type!='existence_constraint'"> In an existence constraint
```

```
      rule it is possible to define one or two table names
    </iso:assert>

    <iso:assert test="(@type='existence_constraint'
     and count(table_name)=2
     and count(table_name)=count(distinct-values(table_name)))
     or @type!='existence_constraint'
     or not(count(table_name)=2)"> Table names have to be distinct
    </iso:assert>
  </iso:rule>
```

– In an *existence constraint* rule, when the attribute `type` for the tag `rule_ec` is equal to `ec_attr`, only one column has to be specified under the tag `rule_ec`:

```
<iso:rule context="rule_ec">
  <iso:assert test="(@type='ec_attr' and count(./column)=1)
   or (@type!='ec_attr')"> An existence constraint rule
    on a single attribute requires only one column under
    the tag rule_ec
  </iso:assert>
</iso:rule>
```

– In an *existence constraint* rule, when the attribute `type` for the tag `rule_ec` is equal to `ec_bidir`, two columns have to be present under the tag `rule_ec`:

```
<iso:rule context="rule_ec">
  <iso:assert test="(@type='ec_bidir' and count(./column)=2)
   or (@type!='ec_bidir')"> An existence constraint rule on
    two attributes requires two columns under the tag rule_ec
  </iso:assert>
</iso:rule>
```

– In an *existence constraint* rule, if some table names are specified under the tag `rule_ec`, the same table names have to be present in the tag `rule_definition`:

```
<iso:rule context="rule_definition">
  <iso:assert test="(@type='existence_constraint' and
   count(./table_name) = count(rule_ec//table_name) and
   count(distinct-values(rule_ec//table_name)) =
```

```
    count(distinct-values(.//table_name)) and
    count(distinct-values(./table_name)) =
    count(distinct-values(.//table_name)))
    or (@type='existence_constraint' and
    count(./table_name)=1 and count(rule_ec//table_name)=0)
    or @type!='existence_constraint'"> Table names under rule_ec
     have to be equals to table names under rule_definition
   </iso:assert>
</iso:rule>
```

– In rules of type *functional dependency*, *order dependency*, and *differential dependency* the column names have to be distinct (we report only the assertion for the functional dependency rule because the others have the same structure):

```
<iso:rule context="rule_fd">
  <iso:assert test="count(.//column_name) =
   count(distinct-values(.//column_name))">
    Column names have to be distinct
  </iso:assert>
</iso:rule>
```

– The previous constraint applies also to an *existence constraint* rule when the attribute `type` for the tag `rule_ec` is equal to `ec_dep` or `ec_bidir`:

```
<iso:rule context="rule_ec">
  <iso:assert test="((@type='ec_dep' or @type='ec_bidir')
   and count(.//column_name) =
   count(distinct-values(.//column_name)))">
    Column names have to be distinct
  </iso:assert>
</iso:rule>
```

– If more than one table is used in a rule of type *check constraint*, the table names have to be distinct:

```
<iso:rule context="rule_definition">
  <iso:assert test="(@type='check_constraint'
   and count(table_name)>1 and count(table_name) =
   count(distinct-values(table_name)))
```

```
    or @type!='check_constraint' or count(table_name)=1">
     Table names have to be distinct
   </iso:assert>
 </iso:rule>
```

– In a *check constraint* rule type, if some table names are specified under the tag `rule_cc`, the same table names have to be present in the tag `rule_definition`:

```
<iso:rule context="rule_definition">
  <iso:assert test="(@type='check_constraint' and
   count(./table_name) = count(rule_cc//table_name) and
   count(distinct-values(./table_name)) =
    count(distinct-values(.//table_name)) and
   count(distinct-values(rule_cc//table_name)) =
    count(distinct-values(.//table_name)))
   or (@type='check_constraint' and
   count(./table_name)=1 and count(rule_cc//table_name)=0)
   or @type!='check_constraint'"> Table names under rule_cc
    have to be equals to table names under rule_definition
  </iso:assert>
</iso:rule>
```

## 4.3 Data Quality measures

In the case study context, in order to quantify the quality of the data provided by Member States, the main measure we introduced is closely connected with the data quality rules. In fact, the used metric is based on the number of records in a dataset satisfying the rules defined for the dataset itself.

Given a set of rules and called respectively:

- $N$ the total number of rules
- $S_r$ the number of records satisfying the rule $r$
- $U_r$ the number of records not satisfying the rule $r$

$$I_{DQ} = \sum_{r=1}^{N} \frac{I_r}{N} \qquad I_r = 1 - \frac{U_r}{S_r + U_r}$$

The index $I_{DQ}$ will assume values in the range [0, 1], with 1 indicating that all the data in the dataset satisfy all the defined rules.

In order to take into account the relevance of the rules, a second version of this type of metric considers also the weight concept.

Given a set of rules and called respectively:

- $N$ the total number of rules
- $w_r$ the weight of the rule $r$
- $S_r$ the number of records satisfying the rule $r$
- $U_r$ the number of records not satisfying the rule $r$

$$I_{DQ_w} = \sum_{r=1}^{N} -\frac{I_r * w_r}{N} \qquad I_r = \frac{U_r}{S_r + U_r}$$

The values of the resulting index $I_{DQ_w}$ will be 0 or a negative number, where 0 is the best value for the index.

Because of the types of rules used in the present work, these measures mainly refer to the accuracy, consistency, and coverage data characteristics. In the case study context, they were used to compare different datasets of different years or subsets of the whole dataset (e.g., data by fleet segment or by Member State).

Other two indexes were used, referring respectively to the completeness of the dataset and to the availability of data in relation to the data submission deadline.

The index referring to the completeness (or incompleteness) of the dataset is computed as follows:

$$I_{coverage} = \sum_{k=1}^{N_M} -\frac{1 * w_k}{N_D}$$

where:

- $N_M$ is the total number of missing values
- $N_D$ is the total number of data values
- $w_k$ is the weight (i.e., relevance) for the missing attribute $k$

The index referring to the availability of data considers the deadline date for the submission of data as the starting date for the life cycle of data and is computed in the following way:

$$\forall\, k \begin{cases} \text{if } \; date_k > date_{deadline} & distance_k = date_k - date_{deadline} \\ \text{otherwise} & distance_k = 0 \end{cases}$$

$$I_{availability} = 1 - \sum_{k=1}^{N_D} \frac{distance_k}{N_D}$$

where $N_D$ is the total number of data values.

CHAPTER 5

# Discovery of Data Quality Rules

In order to explore the case study datasets, we used the open source systems for data mining called Weka (Waikato Environment for Knowledge Analysis) [Hall et al., 2009] and RapidMiner [Mierswa et al., 2006], with particular reference to the following algorithms: Apriori, Tertius, JRip, NNge, and OneR.

Afterwards, to deepen the analysis of the used datasets and with the additional purpose to investigate the dependencies that have been recently proposed in literature in the data cleaning field, we developed some algorithms for discovering the dependencies used as quality rules in our tool.

This chapter, in particular, describes the algorithms implemented to discover CFDs and CFD$^{p}$s in datasets.

## 5.1 Implemented algorithms

The algorithms were implemented using the Java programming language and the PostgreSQL database; they were tested using some of the datasets provided by the UCI Machine Learning Repository [Frank and Asuncion, 2010], such as Iris, Lenses, Wisconsin breast cancer, Yeast, and Zoo datasets.

For this kind of algorithms, the worst case time complexity with respect to the number of attributes is exponential because the number of minimal dependencies – the type of dependencies the algorithms are looking for – can be exponential with respect to the number of attributes.

In order to reduce the complexity, we used an input parameter setting the maximum number of attributes that the target dependencies have to contain. The introduction of this restriction improves the algorithms efficiency for relations containing many attributes.

### 5.1.1 CFD discovery algorithm

As introduced in chapter 3, Conditional Functional Dependencies (CFDs) are Functional Dependencies (FDs) holding on a subset of the tuples of the original relation instance.

The definition of CFDs includes traditional FDs, which are CFDs characterized by tableaux with unnamed variables only, and constant CFDs, which contain in their tableaux only constant values. The most interesting CFDs are those between these extremes, namely, the dependencies containing in their tableaux both constant values and unnamed variables; we refer to them as *variable* or *non-constant* CFDs.

We are looking for this kind of CFDs and, in particular, for non-constant CFDs containing at least one unnamed variable in both the sides of the dependencies – left hand side ($LHS$) and right hand side ($RHS$) – and constant values only in the $LHS$; namely, traditional FDs holding on a subset of the relation instance.

The original definition of CFDs [Fan et al., 2008b] (see chapter 3) was based on the pattern tableau concept, so that CFDs are allowed to have multiple pattern tuples. However, a tableau $T_p$ for a CFD is equivalent to a set of CFD having a single pattern tuple $t_p \in T_p$ [Fan et al., 2011]. For this equivalence, the implemented algorithm is working with CFDs characterized by single pattern tuples; that is, the CFD *normal form* defined in [Fan et al., 2008b] and already mentioned in chapter 3.

Furthermore, to simplify the discovery process, the algorithm manages only CFDs with a single attribute in the RHS, without loss of generality because of the Armstrong decomposition rule: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

More formally, the type of CFDs we are interested in can be written in the form ($LHS \rightarrow RHS$, $t_p$), where:

- $LHS \rightarrow RHS$ is the FD embedded in the CFD;
- $X, T \subset LHS$, $LHS = X \cup T$ and $X \cap T = \emptyset$;
- $RHS$ contains a single attribute $A$;
- $t_p$ is a pattern tuple with attributes in $LHS$ e $RHS$;
- $\forall\ B \in T$ $t_p[B]$ is a constant, $\forall\ Z \in X$ $t_p[Z]$='_', and $t_p[A]$='_'.

In the following, we will refer to the attributes in $T$ (for which constant values are searched) as *target attributes* and to the constants in $t_p[T]$ as *target values*.

Moreover, the embedded FDs that the algorithm looks for are *minimal* and *non-trivial* dependencies.

**Minimal non-trivial dependency.** A dependency $X \rightarrow A$ is not trivial if $A \notin X$; moreover, it is minimal when $A$ does not functionally depend on any proper subset of

$X$, so that each attribute in X is necessary for the dependency to hold (i.e., $X \rightarrow A$ is minimal if, $\forall\ Y \subset X$, $Y \rightarrow A$ does not hold).

In addition, we are particularly interested in the discovery of dependencies *actively* satisfied (versus *vacuously* satisfied) because they are more meaningful in the data quality context. The definition of FDs actively satisfied by a relation can be found in [Mannila and Raiha, 1985].

**Actively satisfied functional dependency.** Given a relation instance $r$ and a functional dependency $X \rightarrow Y$, if there exist distinct tuples $t_1$ and $t_2$ in $r$ such that $t_1[X] = t_2[X]$ and if $t_1[Y] = t_2[Y]\ \forall\ t_1$, $t_2$, then the relation $r$ is said to satisfy the functional dependency *actively*; otherwise, if $t_1$[X] $\neq t_2$[X] $\forall$ distinct $t_1$, $t_2$, then the relation $r$ is said to satisfy the functional dependency *vacuously*.

It is evident that a FD actively satisfied gives more useful information when it is used to evaluate the quality of a dataset.

A last requirement refers to the definition of minimality extended to pattern tuples as defined in [Fan et al., 2011].

**Pattern tuple minimality.** A pattern tuple $t_p$ is minimal, or most general, if none of the constants in the pattern tuple can be "upgraded" to '_'.

Summarizing, we are looking for: *minimal non-trivial* CFDs with the RHS containing a single attribute and with *actively satisfied* embedded FDs; or, in other words, *minimal*, *non-trivial*, *actively satisfied* FDs with the RHS containing a single attribute, holding on the largest subsets of the relation instance.

The basic steps performed by the algorithm are:

- Building the candidates for FDs and CFDs;
- Pruning the number of candidates, when applicable;
- Testing the generated candidates on the relation instance.

A modified version of the algorithm allows the discovery of only frequent CFDs, frequent CFDs being a subset of all the existing CFDs.

**k-frequent CFD.** As defined in [Fan et al., 2011], a CFD is said to be $k$-frequent in a relation $r$ if the support (i.e., the tuples that match the pattern of the CFD) of the dependency is equal or greater than $k$, with $k \geq 1$.

### Candidate generation

Given a FD of the form $LHS \rightarrow Y$ with the attribute $Y$ having all distinct values in the relation instance, then or the FD is only vacuously satisfied (i.e., $LHS$ assumes distinct values for all the tuples) or the FD is not satisfied (i.e., $LHS$ does not have

all distinct values). For this reason, attributes with distinct values in all the tuples can be ignored when building the $RHS$ of a FD candidate.

To build the candidates for FDs, for each attribute $A$, which does not have only constant values or distinct values in all the tuples, the algorithm generates all the dependencies of the form $LHS \rightarrow A$, where $LHS \subseteq R - A$, the minimum size of $LHS$ is 1, and the maximum size of $LHS$ depends on the input parameter. When the size of $LHS$ is equal to 1, the candidate cannot generate any CFDs in the form we are interested in; however, the algorithm uses the candidate to verify the corresponding FD for pruning purposes (as explained later).

To build the $LHS$ of the candidates, the algorithm starts from each single attribute $B \subseteq R - A$ (that does not have only constant values or distinct values in all the tuples), proceeding afterwards in generating larger $LHS$ containing all the combinations of two, three, etc. attributes till the requested maximum number. The generation of the combinations of $LHS$ containing more than two attributes is based on the prefix block approach [Huhtala et al., 1999]: two sets belong to the same prefix block if they have a common prefix of length $l - 1$ (i.e., they differ only for one attribute).

To determine the set of target attributes $T$, the algorithm excludes the attributes that have only constant values or distinct values in all the tuples. Moreover, the values used as constants to be assigned to the target attributes are those that have more than one occurrence in the tuples.

#### Candidate pruning

The approaches implemented in the algorithm to prune the number of candidates to be tested are based on the following criteria:

- When a FD $LHS \rightarrow RHS$ holds, it is not necessary to search for any CFDs $(LHS \rightarrow RHS, t_p)$ having embedded the holding FD.
- If $A_i \dots A_j \rightarrow A_k$, is already verified to be a valid FD, it is not necessary to check the candidate $A_i \dots A_j A_{j+1} \rightarrow A_k$ because it is already known to be a valid FD. This can be demonstrated using the Armstrong axiom of augmentation and the decomposition rule: if $X \rightarrow Y$, then $XZ \rightarrow YZ$, and then $XZ \rightarrow Y$ and $XZ \rightarrow Z$.
- If a CFD $(XT \rightarrow RHS, t_p)$ holds, it is not necessary to check the candidates $(X_2T_2 \rightarrow RHS_2, t_{p_2})$, where $RHS_2 = RHS$, $X \subseteq X_2$, $T \subseteq T_2$ and $\forall\ B \in T$ $t_p[B] = t_{p_2}[B]$.

**The algorithm pseudocode**

The pseudocode of the algorithm developed to discover CFDs is shown in the following.

*Input*: relation $r$ over schema $R$, $maxLHS$ (maximum number of attributes in LHS)

$Set_{FD} := \emptyset$
$Set_{CFD} := \emptyset$
$R_1 :=$ preliminaryPruning($R$)
for each $A_k \in R_1$ do
  $RHS := A_k$
  $R_2 := R_1$ - $A_k$
  $l := 1$
  $L_1 := \{\{A\} | A \in R_2\}$
  testFDCandidatesAndPrune($L_1, RHS$)
  while $l < maxLHS$ do
    $L_{l+1} :=$ generateCandidates($L_l$)
    testCandidatesAndPrune($L_{l+1}, RHS$)
    $l := l + 1$

procedure generateCandidates($L_l$)
  $L_{l+1} := \emptyset$
  for each $K \in$ prefixBlock($L_l$) do
    for each $(Y, Z) \subseteq K, Y \neq Z$ do
      if $\forall A \in X, X \backslash \{A\} \in L_l$ then
        $L_{l+1} := L_{l+1} \cup \{X\}$
  return $L_{l+1}$

procedure testCandidatesAndPrune($L_l, RHS$)
  for each $LHS \in L_l$ do
    if testFDCandidate($LHS, RHS$) then
      $Set_{FD} := Set_{FD} \cup \text{FD}(LHS, RHS)$
      $L_l := L_l - LHS$
    else
      $tl := 1$
      $TL_1 := \{\{A\} | A \in LHS\}$
      while $tl <= (size(LHS) - 1)$ do
        testCFDCandidatesAndPrune($TL_{tl}, LHS, RHS$)
        $TL_{tl+1} :=$ generateCFDCandidates($TL_{tl}$)
        $tl := tl + 1$

procedure testCFDCandidatesAndPrune($TL_{tl}, LHS, RHS$)
  for each $A \in TL_{tl}$ do
    $T := A$
    $X := LHS - T$
    for each $V \in dom(T)$ do
     $t_p[T] := V$
     if not pruneCFDCandidate($T, X, RHS, t_p[T]$) then
      if testCFDCandidate($T, X, RHS, t_p[T]$) then
        $Set_{CFD} := Set_{CFD} \cup \text{CFD}(T, X, RHS, t_p[T])$

procedure generateCFDCandidates($TL_{tl}$)
  $TL_{tl+1} := \emptyset$
  for each $K \in$ prefixBlock($TL_{tl}$) do
    for each $(Y, Z) \subseteq K, Y \neq Z$ do
     if $\forall\ A \in X, X \backslash \{A\} \in TL_{tl}$ then
      $TL_{tl+1} := TL_{tl+1} \cup \{X\}$
  return $TL_{tl+1}$

procedure pruneCFDCandidate($T, X, RHS, t_p[T]$)
  if $\exists\ (X_1T_1 \rightarrow RHS_1,\ t_{p_1}) \in Set_{CFD}$
   and $RHS_1 = RHS$ and $T_1 \subseteq T$ and $X_1 \subseteq X$
   and $\forall\ B \in T_1\ t_{p_1}[B] = t_p[B]$
    return true
  else
    return false

procedure testFDCandidatesAndPrune($L_l, RHS$)
  for each $LHS \in L_l$ do
    if testFDCandidate($LHS, RHS$) then
     $Set_{FD} := Set_{FD} \cup \text{FD}(LHS, RHS)$
     $L_l := L_l - LHS$

procedure preliminaryPruning($R$)
  $R_1 := \emptyset$
  for each $A_i \in R$ do
    $distinctValues :=$ number of distinct $a \in A_i$
    $totalValues := size(A_i)$
    if ($distinctValues > 1$ and $distinctValues < totalValues$) then
     $R_1 := R_1 \cup A_i$
  return $R_1$

### 5.1.2 CFD$^p$ discovery algorithm

Among the dependencies recently proposed in literature (already mentioned in chapter 3), the CFD$^p$s introduced in [Chen et al., 2009] seem to be particularly useful, both for data quality assessment and for data cleaning purposes. In spite of this, for the discovery of this type of dependency we have not found any algorithms published in literature.

Searching for CFD$^p$s, we consider only numerical attributes and as comparison operators those considered in [Chen et al., 2009] (i.e., $\neq$, $<$, $>$, $\leq$, $\geq$).

The algorithm is characterized by the same general structure already described for the CFDs discovery algorithm. In order to determine the candidates for the dependencies, the same steps are executed; the two algorithms differentiate, after the selection of a candidate, in the way the conditions for the target attributes are determined. After the general candidate $XT \rightarrow RHS$ for a CFD$^p$ has been generated, the algorithm performs the following steps:

1. For the attributes in the set $X$ and in $RHS$, (i.e., all the attributes not in the target set $T$), it produces the sets containing only tuples with identical values;
2. It generates partitions of the computed sets: each partition contains only sets having different values for the attributes in the set $X$, thus assuring that all the tuples in each partition satisfy the embedded FD;
3. For each partition, it iterates on the following steps (the number of iterations depends on the number of partitions, but it can be limited by the user through an input parameter):
   (a) It merges the sets contained in the partition;
   (b) It finds a range containing all the values for each attribute in the set $T$ (using the min and max values);
   (c) From the range of values for the target attributes in the set $T$, it deletes the values corresponding to the records contained in the sets that were not used in the merging step;
   (d) It examines the resulting set of values for the target attributes in the set $T$ and decides if the result has to be kept or refused.

We experimented also with the development of an algorithm for discovering CFD$^p$s containing only the inequality operator (i.e., $\neq$). In this case, the used approach is based on the discovery of dependencies that do not hold; then, these values are used as negative conditions if the set of values for the target attributes is small (the maximum size of this set can be assigned through an input parameter).

## 5.2 Related works

The discovery of dependencies has attracted many research interests from the communities of database design, machine learning, and knowledge discovery since early 1980s.

Two typical types of dependencies are often involved in the discovery: FDs and inclusion dependencies. FDs represent value consistencies between two sets of attributes; in contrast, inclusion dependencies represents value reference relationships between two sets of attributes. Moreover, some algorithms for discovering CFDs have been recently proposed in literature.

A recent work [Liu et al., 2012] reviews the methods proposed in literature to discover FDs, approximate FDs, CFDs, and inclusion dependencies in relational databases.

### 5.2.1 FD discovery methods

The methods proposed in literature to discover FDs from data can be classified as top-down and bottom-up [Liu et al., 2012].

The top-down methods start with generating candidates for FDs level-by-level, from short LHS to long LHS; then they verify if the candidates are valid dependencies on the relation instance. The bottom-up methods, on the other hand, start with comparing tuples to get agree sets or difference sets; then they generate candidates for FDs and check them against the agree sets or difference sets for satisfaction.

The main difference between the two types of methods is that the first one checks the candidates against the relation instance for satisfaction, while the second one checks the candidates against the computed agree sets or difference sets.

Examples of well-known algorithms implementing a top-down method are TANE [Huhtala et al., 1999] and FD_Mine [Yao et al., 2002, Yao and Hamilton, 2008], while Fast-FD [Wyss et al., 2001] and Dep-miner [Lopes et al., 2000] are algorithms adopting a bottom-up method.

On the basis of these definitions, the algorithms we developed to discover CFDs and CFD$^{\text{P}}$s can be classified as top-down methods.

### 5.2.2 CFD discovery algorithms

For the discovery of general CFDs the following algorithms have been proposed:

– Chiang and Miller [2008] present an algorithm based on the attribute lattice search strategy for discovering both constant and non-constant CFDs.

– Fast-CFD [Fan et al., 2011] is inspired by the Fast-FD algorithm.

– CTANE [Fan et al., 2011] extends the TANE algorithm.

– CFD-Mine [Aqel et al., 2012] is also based on an extension of the TANE algorithm.

Moreover, two algorithms for the discovery of only constant CFDs have been recently proposed:

– CFDMiner [Fan et al., 2011] is based on techniques for mining closed item sets and finds a canonical cover of k-frequent minimal constant CFDs.

– In [Li et al., 2012] new criteria to further prune the search space used by CFDMiner to discover the minimal set of CFDs are proposed.

For the discovery of CFD$^{\text{p}}$s, to date and to our knowledge, there are no published algorithms.

CHAPTER 6

# Data Provenance

In many scientific fields (e.g., biology, chemistry, physics) large amounts of data are collected and stored in various data repositories.

Data stored in *curated databases*[1] are considered trustworthy because they are under centralized control; however, data stored in curated repositories are normally collected from different sources. Because information sources, or different parts of a single large source, may vary widely in terms of quality, researchers can be interested in having information about both data sources and transformations applied to data.

On the other hand, on the World Wide Web data are often made available with no centralized control over their integrity. In this case, providing provenance and other context information can help end users in judging if data are reliable.

In general, the term data provenance refers to the process of tracing and recording the origins of data. The provenance of a data item generally includes information about the source data items and the processes that lead to its creation and current representation [Glavic and Dittrich, 2007].

Provenance in terms of origin and history of the changes of a dataset enables its users to evaluate its suitability for a particular application; thus, provenance information can play an important role in the data quality context.

Provenance has been extensively studied in the context of relational databases [Tan, 2007] and for workflow management systems [Davidson et al., 2007].

In relational databases, provenance describes relationships between data in the source and in the output by propagating fine-grained annotations or algebraic expressions from the input to the output. Different models of provenance for database queries have been proposed; however, the main notions of database provenance are

[1] The term *curated database* is normally used for databases populated and updated by domain experts through the consultation, verification, and aggregation of existing sources [Buneman et al., 2008].

called why provenance, how provenance, and where provenance. They describe the relationships between data in the source and in the output by showing why an output record was produced, or describing in detail how an output record was produced, or showing data sources from where output data came. A survey about these three main notions of database provenance can be found in [Cheney et al., 2007].

Provenance support has been recognized as an important added value in scientific workflow systems. For this reason, provenance models have been developed for several workflow management systems, such as Chimera [Foster et al., 2002], Karma [Simmhan et al., 2008], Kepler [Anand et al., 2009], Taverna [Missier et al., 2008], and ZOOM [Biton et al., 2008]. Data provenance in workflows is generally captured as a set of dependencies between data objects; however, the amount of information recorded for provenance varies among the systems.

The target of most of the researches in the data provenance field has been the implementation of concrete systems in the context of either specific domains or technologies. Recently, however, the Open Provenance Model [Moreau et al., 2011] has been proposed as an exchange format for representing provenance graphs independently from the used technology or domain.

## 6.1 The Open Provenance Model

Existing provenance systems do not share a common data model for provenance. To promote and facilitate interoperability among heterogeneous provenance systems the Open Provenance Model (OPM) [Moreau et al., 2007] was proposed in 2007; since then, it influenced the activities of the data provenance community.

OPM was designed to allow provenance information to be exchanged between different and heterogeneous systems by means of a compatibility layer based on a shared provenance model. Moreover, OPM defines a core set of rules which can be used to identify the valid inferences that can be made on provenance representation.

Provenance of objects is represented by an annotated graph, which is a directed acyclic graph expressing casual dependencies among entities enriched with annotations capturing further information pertaining to execution.

The graph is based on three kinds of nodes called *artifact*, *process*, and *agent*:

- An *artifact* is an immutable piece of state which may have a physical embodiment in a physical object or a digital representation in a computer system.
- A *process* is an action (or series of actions) performed on or caused by artifacts and resulting in new artifacts.

– An *agent* is an entity acting as a catalyst of a process, enabling, facilitating, controlling, or affecting its execution.

OPM aims at capturing the causal dependencies among artifacts, processes, and agents. A causal relationship is represented by an arc that denotes the presence of a causal dependency between the source of the arc (the effect) and the destination of the arc (the cause).

The following five causal relationships are recognized in OPM:

1. A process used an artifact: a "used" edge from a process to an artifact is a causal relationship indicating that the process required the availability of the artifact to be able to complete its execution. When several artifacts are connected to a same process by "used" edges, all of them were required for the process to complete.

2. An artifact was generated by a process: a "was generated by" edge from an artifact to a process is a causal relationship meaning that the process was required to initiate its execution before the artifact can be generated. When several artifacts are connected to a same process by multiple "was generated by" edges, the process had to have begun, for all of them to be generated.

3. A process was triggered by (or informed by) another process: a "was triggered by" edge from the process P2 to the process P1 is a causal dependency indicating that the start of the process P1 was required for P2 to be able to complete. The relationship P2 "was triggered by" P1 only expresses a necessary condition (not a sufficient one).

4. An artifact was derived from another artifact: a "was derived from" edge from the artifact A2 to the artifact A1 is a causal relationship meaning that the artifact A1 needs to have been generated for A2 to be generated.

5. A process controlled by an agent: a "was controlled by" edge from a process P to an agent Ag is a causal dependency that indicates that the start and the end of the process P was controlled by the agent Ag.

In the graphical notation for provenance graphs, artifacts are represented by ellipses, processes are represented by rectangles, and agents are represented by octagons (figure 6.1); moreover, in the graphs, sources are effects and destinations are causes.

In OPM the concept of *role* designates the function of an artifact or agent in a process. Roles are constituents of "used", "was generated by", and "was controlled

by" edges; their aim is to distinguish the nature of the dependency when multiple such edges are connected to a same process:

- An artifact may be used by more than one process, possibly for different purposes, and a process may use or generate more than one artifact; each "used" and "was generated by" relation may be distinguished by a role with respect to that process.
- An agent may control more than one process; in this case, the processes may be distinguished by the role associated with the "was controlled by" relation.
- A process may be controlled by more than one agent; in this case, each agent may have a distinct controlling function, which would be distinguished by roles associated with the "was controlled by" relations.

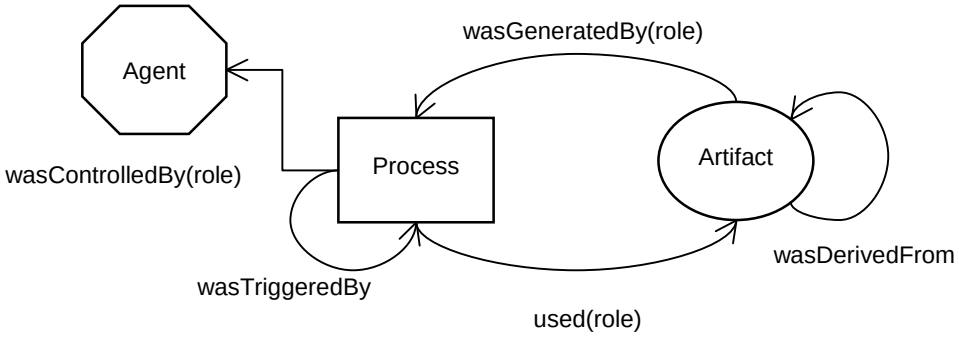



**Figure 6.1:** Basic concepts and relationships defined in OPM

Therefore, a provenance graph is defined as a directed graph, whose nodes are artifacts, processes, and agents, and whose edges belong to one of the following categories: *used(role)*, *wasGeneratedBy(role)*, *wasTriggeredBy* (or *wasInformedBy*), *wasDerivedFrom*, and *wasControlledBy(role)*.

Causality is not the only relationship among provenance entities, it is useful to record other relevant domain-specific relationships; in OPM this can be done using annotations. Thus, by means of annotations, edges can be further subtyped from the five defined categories.

Provenance information about an artifact could be expressed at different levels of abstraction or from different viewpoints. The OPM specification introduces the concept of *account* to represent a description at some level of detail as provided by one or more observers.

### Completion rules

The causal dependencies *wasTriggeredBy* and *wasDerivedFrom* are summary edges for a process view, where an intermediary artifact is unknown, and a data view, where an intermediary process is unknown, respectively. In [Moreau et al., 2011] completion rules for these causal dependencies were introduced.

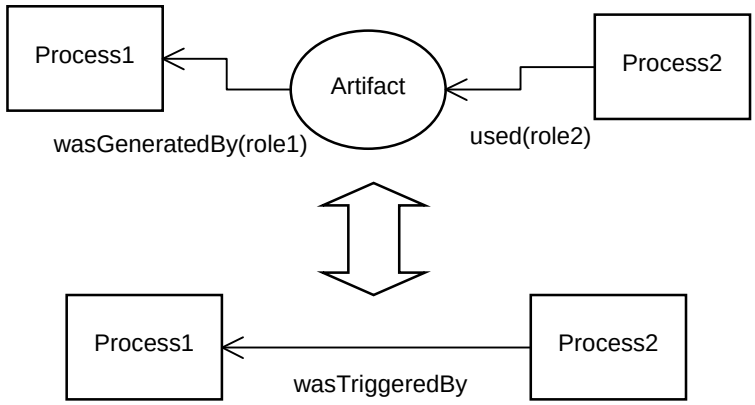



**Figure 6.2:** Artifact introduction and elimination

The *wasTriggeredBy* edge in OPM can be inferred from the existence of a *wasGeneratedBy* edge and a *used* edge. If one process generated an artifact that was used by another process, then the latter "was triggered by" the former (figure 6.2).

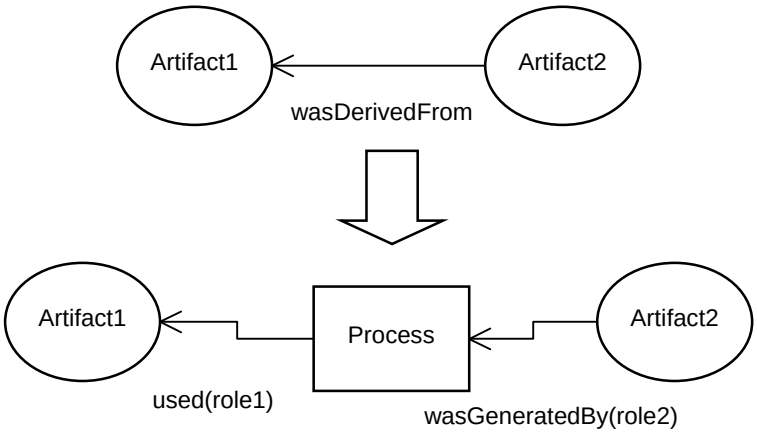



**Figure 6.3:** Process introduction

In a similar way, a *wasDerivedFrom* edge hides the presence of an intermediary process (figure 6.3). However, the converse rule does not hold because, without any

internal knowledge of the process, it is impossible to infer if there is or not an actual dependency between the used artifact and the one that was generated. The fact that a process used an artifact and generated another does not imply that the latter was derived from the former; therefore, such a relationship needs to be asserted explicitly.

### Multi-step inferences

With the purpose of expressing queries or inferences about provenance graphs, in [Moreau et al., 2011] other four relationships, which are multi-step versions of existing OPM relationships, were added to the model.

The first one is called multi-step *wasDerivedFrom* and indicates that the artifact A1 was derived from the artifact A2 (possibly using multiple steps), written as A1→*A2, if A1 "was derived from" an artifact that was A2 or that was itself derived from A2 (possibly using multiple steps). In other words, it is the transitive closure of the edge *wasDerivedFrom*, and it expresses that artifact A1 had an influence on artifact A2.

The other relationships are denominated secondary multi-step edges:

1. Process P "used" artifact A (possibly using multiple steps), written P→*A, if P used an artifact that was A or was derived from A (possibly using multiple steps).

2. Artifact A "was generated by" process P (possibly using multiple steps), written A→*P, if A was generated by P or was derived from an artifact (possibly using multiple steps) that was generated by P.

3. Process P2 "was triggered by" process P1 (possibly using multiple steps), written P2→*P1, if P2 used an artifact that was generated by P1 or that was derived from another artifact (possibly using multiple steps) generated by P1.

### Temporal constraints

In OPM graphs, edges can optionally be annotated with time information.

For the *used* and *wasGeneratedBy* edges, the timestamp indicates that the associated artifact was known to be generated or used at a given time; for the *wasDerivedFrom* edge, the timestamp indicates when the source artifact was used; for the *wasTriggeredBy* edge, the timestamp marks the time when the communicated artifact was used by the edge source; finally, for the *wasControlledBy* edge, two optional timestamps are allowed marking when the process was known to be started or terminated, respectively.

When time is expressed in an OPM graph, there are some time constraints to be considered. In particular, the following "happened before" relationships, which are displayed in figure 6.4, must be satisfied:

- an artifact must exist before it is being used (T1 $\leq$ T3 and T4 $\leq$ T6);
- if an artifact is used by a process, it will actually be used after the start of the process (T2 $\leq$ T3) and before the end of the process (T3 $\leq$ T5);
- a process generates artifacts before its end (T4 $\leq$ T5), and a process start precedes its generation of artifacts (T2 $\leq$ T4) and its end (T2 $\leq$ T5).

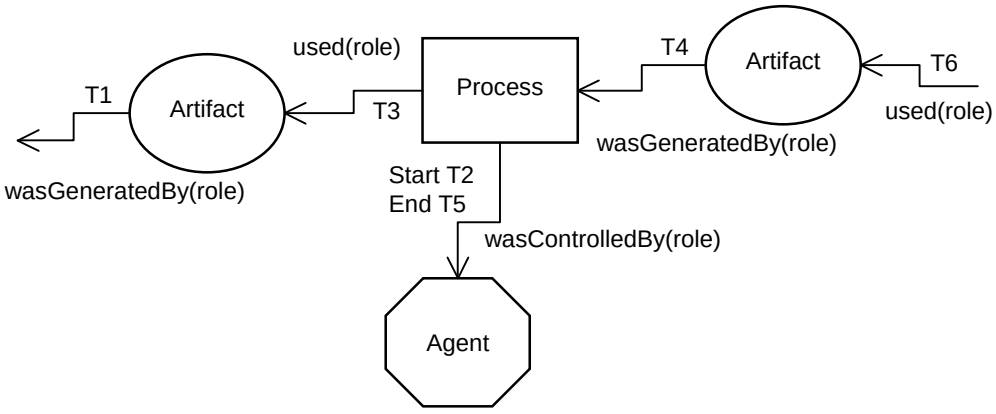



**Figure 6.4:** Time constraints in OPM

### Definitions of legal OPM graph

Two definitions of legal OPM graph have been proposed in [Moreau et al., 2011] and [Kwasnikowska et al., 2010], respectively.

In the specification of the version 1.1 [Moreau et al., 2011] the following definition of legal OPM graph has been introduced: within an account, an artifact has to be generated by a single process (i.e., at most one *wasGeneratedBy* edge per artifact must exist) to avoid conflicts about the artifact origin; moreover, even if cycles can be expressed in the syntax of OPM, to be legal – within an account – the graph has to be acyclic, so that it accurately expresses causal dependencies between processes and artifacts; by extension, an OPM graph is said to be legal if all its account views are legal.

In [Kwasnikowska et al., 2010] a temporal semantics for OPM has been defined in terms of a set of ordering constraints between time-points associated with OPM

constructs (i.e., the beginning of a process, the ending of a process, the instant a process uses an artifact, and the moment a process creates an artifact). In this OPM formalization, the edges used in the graphs are further categorized into precise and imprecise edges (except the *wasControlledBy* edge, which is not considered because the agent node is not mentioned). Precise edges are syntactically marked by the presence of roles to characterize the nature of the relationship between the source and the destination of the edge; by contrast, imprecise edges do not have roles and represent incomplete information. Moreover, a role can be assigned also to the *wasDerivedFrom* edge, while in the specification of the version 1.1 this possibility was not considered.

Therefore, the definition of legal OPM graph has been modified as follows: an OPM graph is called legal if: (1) for each artifact A there is at most one precise *wasGeneratedBy* edge, and (2) for each precise *wasDerivedFrom* edge with role $r$ connecting A to the artifact B, there is a process P connected to B with a precise *used* edge characterized by the same role $r$ and connected to A through a precise *wasGeneratedBy* edge. The configuration described in (2) is called a *use-generate-derive triangle*.

### Graph serialization

OPM is an abstract data model to represent past execution and does not specify the internal representations or protocols that systems have to adopt to store provenance information in repositories nor to query provenance repositories. However, an OPM graph can be serialized in different formats, such as OPM XML Schema and OPM RDF Schema, and in these formats it can be used to export from and to upload to existing systems information about data provenance.

### An OPM graph for the case study

In figure 6.5 a partial OPM graph for the case study is shown (in the graph the roles on the edges are not displayed; moreover, the names of the edges, except the one for the *used* relation, are shortened in the following way: WGB for *wasGeneratedBy*, WDF for *wasDerivedFrom*, WTB for *wasTriggeredBy*, and WCB for *wasControlledBy*; moreover, the *wasDerivedFrom* edges are drawn with dashed lines).

The OPM graph represents the main activities involved in the data collection process from the point of view of both JRC and Member States. Each Member State collects row data through the actions of national public or private institutes; afterwards, the collected data are aggregated at national level. For each Member State a national correspondent coordinates the data collection activities and sends

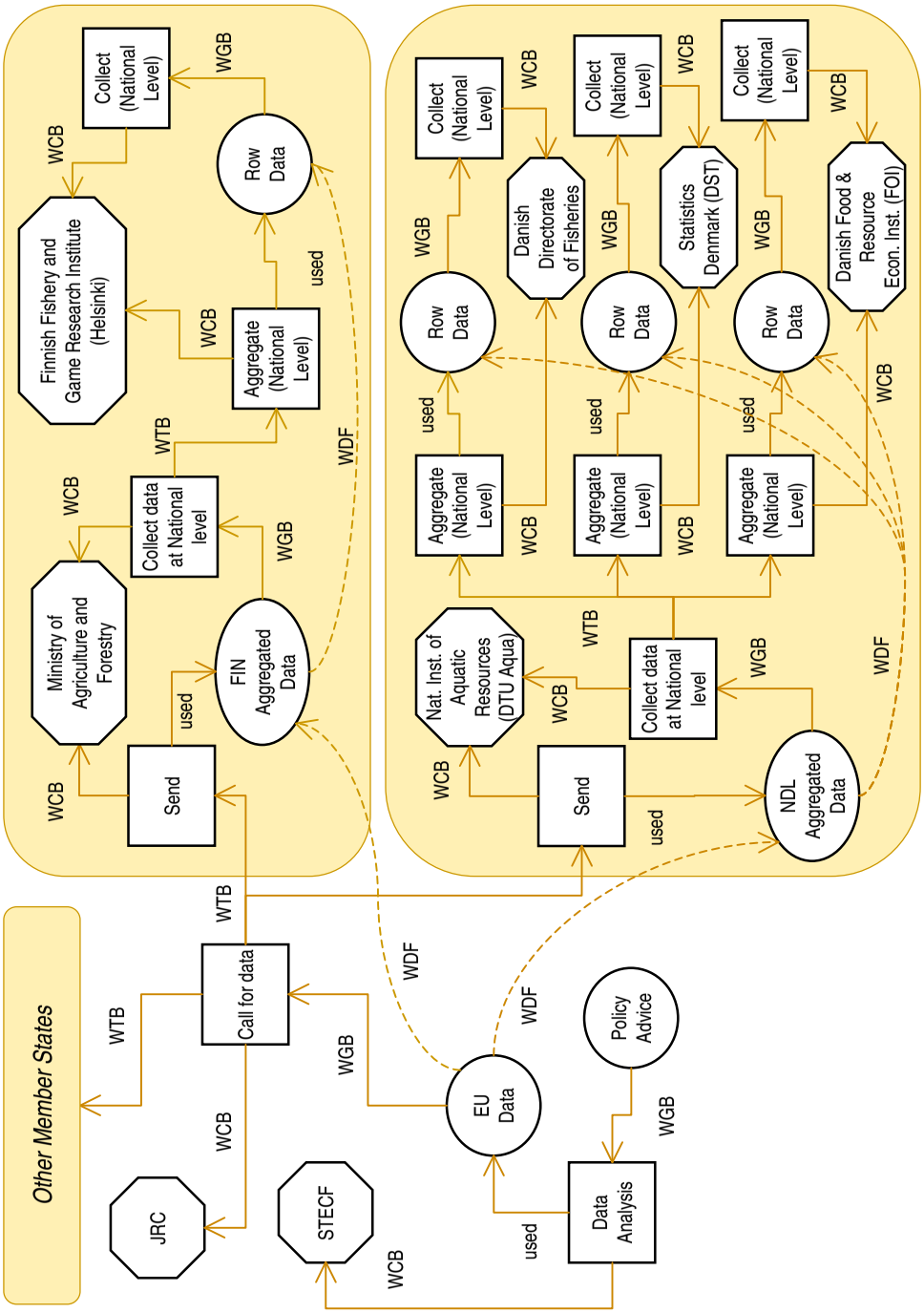



**Figure 6.5:** A partial OPM graph for the case study

the data to JRC on the basis of requests coming from the European Commission. Finally, JRC receives and maintains the data that are then at disposal of the experts.

The graph in figure 6.5 is referring only to two Member States (Finland and Denmark); similar subgraphs are replicated for all the other Member States.

## 6.2 Comments and extension proposal

### Legal graph definitions

As already mentioned in the previous paragraph, two different definitions of legal OPM graph were proposed in [Moreau et al., 2011] and in [Kwasnikowska et al., 2010], respectively.

We observed that in the formalization given in [Kwasnikowska et al., 2010], it is not explicitly mentioned the case in which an artifact, which has imprecise *wasGeneratedBy* edges, does not have any precise *wasGeneratedBy* edges (i.e., with a role specified for it). Thus, there are no indication about which are the constraints that have to be applied in a case like this to obtain a legal OPM account.

In the present work, to be able to deal with OPM graphs showing cases like the mentioned one, when we utilize the definition given in [Kwasnikowska et al., 2010] we assume that for artifacts with no precise *wasGeneratedBy* edges the legal definition given in the OPM specification version 1.1 has to be applied.

### Agents and artifacts

The notion of agent, as recognized also by the authors of the OPM model, is not well defined yet.

In order to better exploit the concept of agent, we propose to add, between an artifact and an agent, a relation implied by a couple of existing edges: a *wasControlledBy* edge between a process P and an agent Ag, and a *wasGeneratedBy* edge between an artifact A and the same process P. An example of this relation between an artifact and an agent is shown in figure 6.6, where the derived edge is drawn with a dashed line.

It is also possible to have a multi-step version of this inferred relation, written as A→*Ag, meaning that artifact A "was controlled by" agent Ag (possibly using multiple steps), if A was generated by a process P controlled by the agent Ag or was derived from an artifact (possibly using multiple steps) that was generated by a process P controlled by the agent Ag. Figure 6.7 shows an example of the proposed multi-step edge between an artifact and an agent, where the multi-step edge is drawn with a dotted-dashed line.

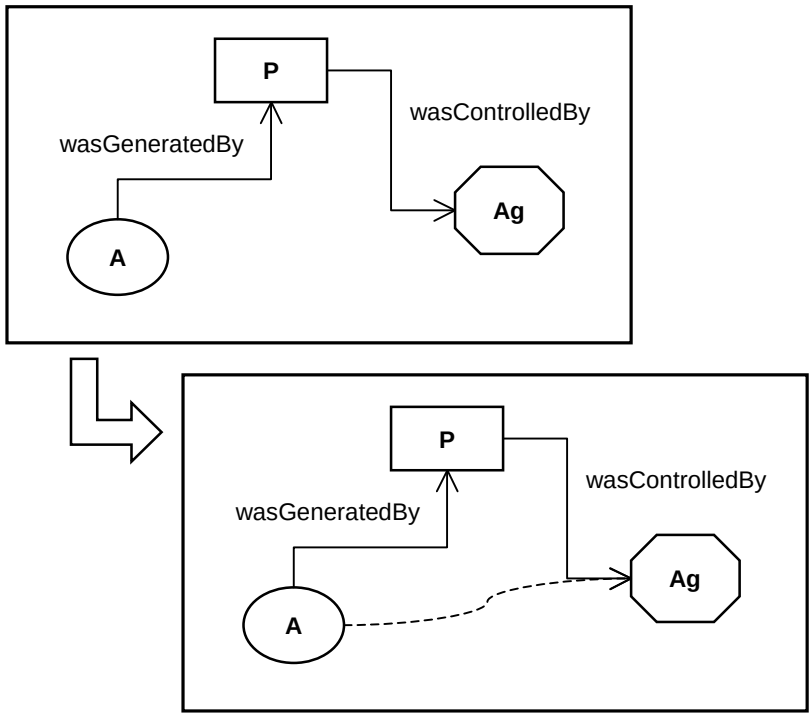



**Figure 6.6:** An inferred relation between an artifact and an agent

Both the inferred relation and the multi-step relation based on it can be used in answering queries about artifacts dependencies on agents.

### Subtypes for the *wasDerivedFrom* edge

In some contexts, it could be useful to distinguish among transformed data and copied data; namely, among artifacts that were derived from other artifacts through modifications or through a copy of the input. This can be obtained introducing in the OPM graph subtypes for the *wasDerivedFrom* edge, such as *wasTransformFrom* and *wasCopiedFrom* edges.

In the current version of the model, this target can be reached using annotations. For example, we can define an annotation containing in the property the label *edgeSubType* and in the value one of the two labels: *transformedFrom* or *copyOf*. It could also be established that the absence of the annotation refers by default to transformed data and only for copied data the annotation is present.

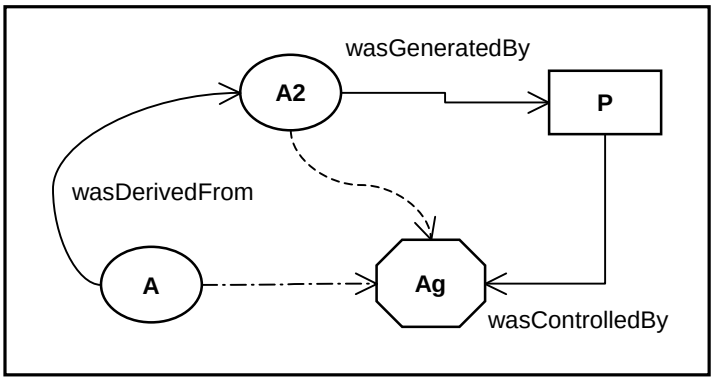



**Figure 6.7:** The multi-step edge artifact-agent

## 6.3 W3C Provenance Working Group

The authors that proposed the Open Provenance Model are currently participating in the activities of the W3C[2] Provenance Working Group.

The mission of the W3C Provenance Working Group is to support the widespread publication and use of provenance information of Web documents, data and resources. In particular, the W3C Provenance Working Group is working at a family of specifications, called PROV, with the target to help in defining how to interchange provenance information.

The activities of the Working Group are based on an extensive review and roadmap developed by a prior incubator group. As indicated in the Incubator Group's report [Gil et al., 2010], many provenance models exist with significantly different expressiveness and different assumptions about the systems they are embedded in. It does not seem realistic today that a single way of representing and collecting data provenance could be adopted internally by all systems. For this reason, the proposal of the W3C Provenance Incubator Group has been, with a pragmatic approach, to consider a core provenance language with an extension mechanism that allows domain and application specific representations of provenance to be translated into such a language and exchanged among systems. Heterogeneous systems can then export their provenance models into such a core language, and different applications can import it and reason over it.

The work of the W3C Provenance Incubator Group finished in 2010, and currently the W3C Provenance Working Group is working – on the directions indicated in the final report produced by the Incubator Group [Gil et al., 2010] – with the

[2]http://www.w3.org

objective to define a provenance interchange language and methods to publish and access data provenance using that language.

On July 2012 the W3C Provenance Working Group published the Last Call Working Draft of the *PROV Data Model* for provenance (PROV-DM), a generic data model for provenance. PROV-DM is a domain-agnostic model with extensibility capabilities allowing further domain-specific and application-specific extensions to be defined.

PROV-DM distinguishes core structures, forming the essence of provenance, from extended structures allowing more specific uses of provenance. At its core, provenance describes the use and production of entities by activities, which may be influenced in various ways by agents. PROV-DM is based on three types of nodes – called *entity*, *activity*, and *agent* – and on seven relations: *wasGeneratedBy*, *used*, *wasInformedBy*, *wasDerivedFrom*, *wasAttributedTo*, *wasAssociatedWith*, and *actedOnBehalfOf*.

The PROV-DM core structure presents many similarities with the Open Provenance Model. They share the same types of nodes: in OPM entities are called artifacts, activities are called processes, and agents have the same name. Concerning the relations provided in the models, PROV-DM is richer than OPM, and thus it has a higher expressiveness. Five of the seven relations present in PROV-DM correspond to the OPM edges. The *wasGeneratedBy*, *used*, and *wasDerivedFrom* relations have same names and same meanings in both the models, while the relations called *wasInformedBy* and *wasAssociatedWith* correspond respectively to the *wasTriggeredBy* and *wasControlledBy* edges in OPM. Moreover, PROV-DM provides two more relations called *wasAttributedTo* and *actedOnBehalfOf*: the *wasAttributedTo* relation can exist between an entity and an agent, and *actedOnBehalfOf* models a relation between two agents.

CHAPTER 7

# OPM Graph Design

By graph design method we mean a set of transformation rules to be applied in order to derive a correct graph, with respect to the initial specifications (e.g., a natural language description).

Design methods and methodologies are widely used both in database and workflow systems. Dealing with data and being based on the workflow concept, also the OPM model can benefit from the exploitation of a design method.

In this chapter, we describe a method for the design of OPM graphs. In the proposed design approach, two phases can be identified: a conceptual phase and a physical phase. In the conceptual phase, concepts and their relationships are identified and mapped into OPM elements; while the physical phase refers to the serialization of the graph into the corresponding XML structure.

In the presentation of the method, we focus on the drawing of an OPM graph containing a single account, even if more accounts can be design together.

It is important to be aware that starting from the description of a dataset and of the workflow generating it, it is possible to produce different OPM graphs.

As already mentioned previously, two different definitions of legal OPM graph have been proposed (in [Moreau et al., 2011] and in [Kwasnikowska et al., 2010], respectively). Being the first definition part of the current reference specification of the model and the second a better interpretation of the semantics of the "was derived from" relation, in the following we will refer to both of them specifying, when necessary, the differences in adopting one or the other.

## 7.1 Entity clusters and abstract subgraphs

In the design of an OPM graph containing several entities, two concepts can be useful: entity cluster and abstract subgraph.

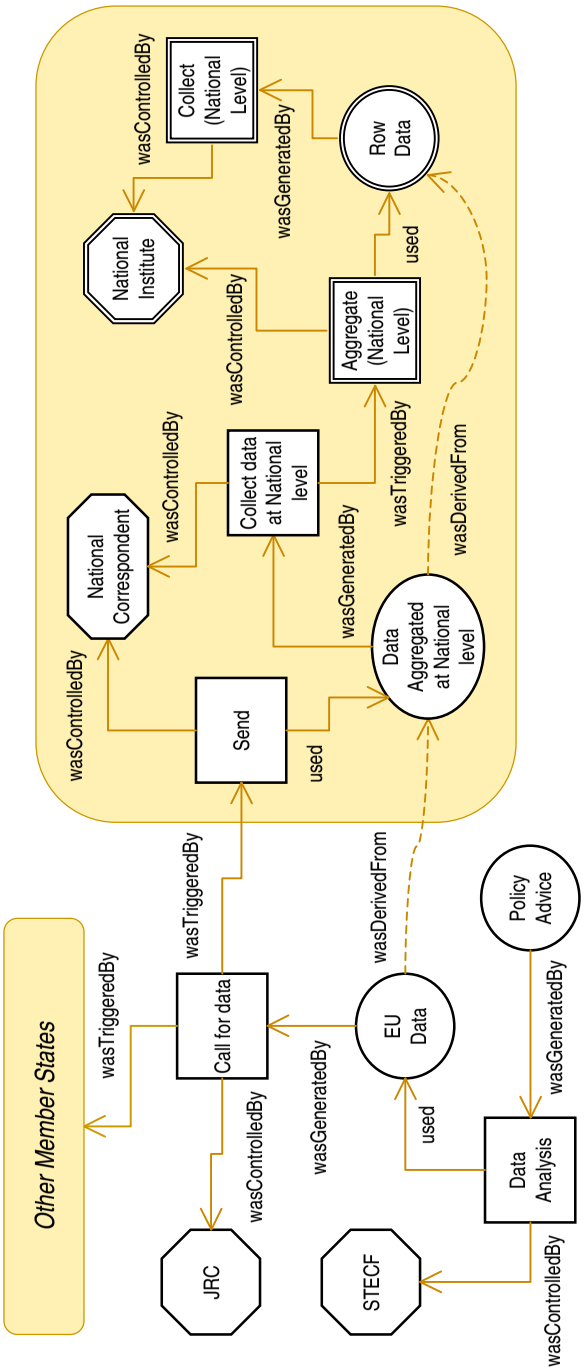



**Figure 7.1:** An example of abstract subgraph from the case study

An entity cluster [Hoffer et al., 2007] represents a set of entities and associated relationships grouped into a single abstract entity. In an OPM graph, in particular, an abstract entity is represented by an abstract node, which can be an artifact, a process or an agent. The idea of entity cluster has been demonstrated to be very useful in the Entity-Relationship model design [Elmasri and Navathe, 2000] when working with large graphs; details can be temporarily ignored to concentrate on the main entities and relationships, approaching the modeling with a top-down view.

An abstract subgraph is an outline of the provenance graph structure without the instance information about actual entities. The use of an abstract subgraph can be useful when the same substructure is present in an OPM graph more than once. An example of an abstract subgraph in the case study context is shown in figure 7.1; in this example, the abstract subgraph (enclosed in the shaded box) represents a portion of the graph that has to be replicated for each Member State. Another abstract subgraph will be used in the example shown later in this chapter (see figure 7.11).

## 7.2 Starting steps: nodes and edges

An analysis of the life cycle of data, or of the portion of the data life cycle of interest, is the prerequisite step in the design of an OPM graph. The main result of this analysis is the identification of the actors and processes that are involved in the creation and transformation of the data.

During the analysis phase, an important preliminary decision has to be taken about the data granularity that will be used in the model.

Afterwards, to draw the graph it is necessary to start identifying the entities that will be inserted in the graph itself. Having chosen a top-down approach, the main data and, therefore, the main artifacts to be used in the model have to be identified. If one of the identified artifacts refers to more than one entity, it can be represented by an entity cluster, and it can be useful drawing it with a double line in order not to forget the need to expand its description in the next steps of the design.

The identification of the participants is the next step, meaning the identification of the agents that will be inserted in the graph.

Having chosen the nodes for artifacts and agents, it is time to consider the processes. In the next steps of the graph design, two tables can help (figure 7.2). The first table lists the identified artifacts and for each artifact: the processes generating and using it, the artifacts on which it depends, and the artifacts that are used as sources for its generation. If there exist artifacts that are generated by more than one process, it is useful to flag them.

| Artifact | Generated by Processes | Used by Processes | Dependent on Artifacts | Sources for Artifacts | Flagged |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

| Process | Generating Artifacts | Using Artifacts | Input/Output dependecies | Dependent on Processes | Controlled by Agents |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

**Figure 7.2:** Tables for OPM entities

The second table contains a list of processes with the artifacts used and generated by each process, the specification of existing dependencies among used and produced artifacts, and the processes on which each process depends. In this table, it is also useful to specify the agents that control each process. If in the first table some artifacts are flagged as being generated by more than one process, the corresponding processes need to be marked as well in the second table.

The tables filling procedure is typically an iterative exercise. The comparison of the content of the two tables can help in identifying inconsistencies among the provided information. For example, it can suggest that in the first table some artifacts are missing or that some not yet considered dependency relations among artifacts can exist. Furthermore, from the data provided in these tables, it is easy to derive the types of the edges connecting the nodes.

Working with the version 1.1 of the OPM specification, in order to draw the graph it is possible to proceed as follows: for each artifact listed in the first table, we can draw (1) the artifact itself, (2) the artifacts on which it depends connected through *wasDerivedFrom* edges, (3) the processes that use the artifact and the *used* edges connecting them, and (4) the processes that generate the artifact with the corresponding *wasGeneratedBy* edges. Only during the step (4) it is necessary to consider if the current artifact was flagged in the first table, namely, if the artifact is generated by more than one process. This can happen, for example, when an artifact derives from the joining of the results of more than one process. In this case, to comply with the legal graph definition, it is necessary to introduce an extra process, even if it was not explicitly identified during the workflow analysis, representing the merging step; figure 7.3 exemplifies how a legal graph can be obtained when an artifact A is generated by more than one process. When all the artifacts listed in the

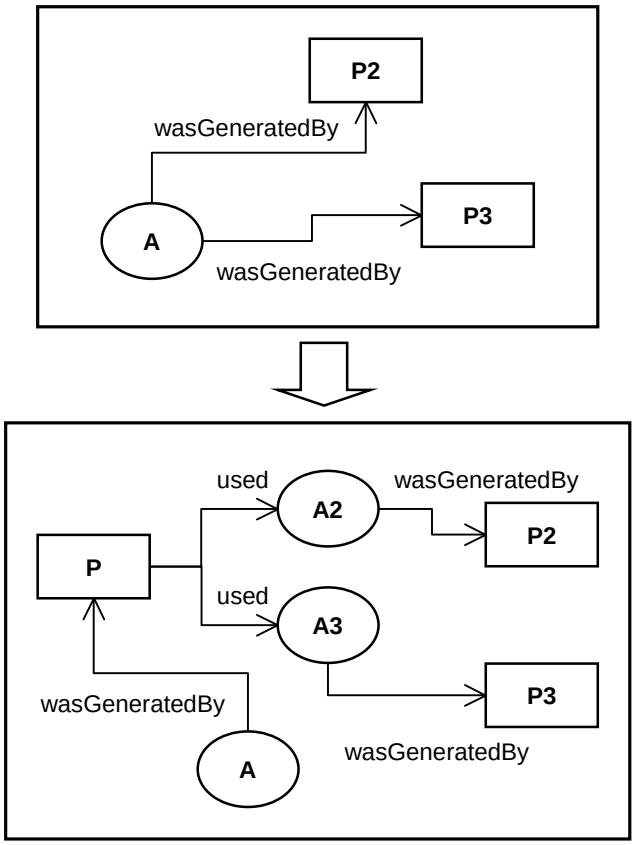



**Figure 7.3:** An example of artifact generated by more than one process

first table are drawn, we can check in the second table if all the listed processes are present in the graph. Referring to the second table, for each process we verify if the cell in the column "dependent on processes" is not empty; for each process present in this cell, we can draw a *wasTriggeredBy* edge. Lastly, the agent nodes have to be inserted and connected through a *wasControlledBy* edge to the corresponding processes as indicated in the second table.

Working with the legal graph definition provided in [Kwasnikowska et al., 2010], further characteristics of the OPM graph have to be considered.

In this case, the model considers both precise and imprecise edges, so in order to distinguish them, it is necessary to explicitly mention in the tables the role associated to each relation. Having the role information, it is possible to verify the compliance with the first part of the definition identifying in the first table the artifacts that are generated by more than one process through a precise edge. Afterwards, in order to verify the compliance with the second part of the definition, we can proceed

as follows. For each artifact A1 contained in the table, we consider the column "dependent on artifacts": for each artifact A2 with an associated role $r$ mentioned in this column, we check if a use-generate-derive triangle exists. First of all, in the row for the artifact A1, a process P with an associated role has to be present in the column "generated by processes" (i.e., referring to a precise *wasGeneratedBy* edge); moreover, in the row for the artifact A2, the same process P with role $r$ has to be present in the column "used by processes".

## 7.3 Checking for cycles

An OPM graph is defined to be acyclic for each account, so it is important to check the presence of cycles referring in particular to the *wasDerivedFrom* edges.

If the presence of a cycle is detected, the graph has to be modified. Firstly, the analysis of the data life cycle has to be reconsidered to identify incorrect edges. However, it is also possible that all the identified edges are correctly representing the relationships among artifacts and processes; for example, when the result of a process, or of a sequence of processes, is an input for the same process. In this case, in order to remove the cycle, it is necessary to decide if it is possible to eliminate the feedback edge.

The presence of cycles can be also verified after the serialization of the graph in the corresponding XML format, as explained later in the chapter.

## 7.4 Redundancy check

A "was triggered by" relation between two processes can be inferred, using one of the completion rules defined for the model (see chapter 6), from the existence of an artifact linked to one of the processes through a *wasGeneratedBy* edge and to the second process through a *used* edge. For this reason, when in an OPM graph two processes are already connected by means of an artifact, the *wasTriggeredBy* edge can be eliminated.

At the contrary, the presence of a *wasDerivedFrom* edge between two artifacts with, at the same time, a process using one of the artifacts and generating the other cannot be regarded as a redundancy. In fact, a *wasDerivedFrom* edge cannot be derived from the mere existence of a process connecting two artifacts; it is necessary (as explained in the description of the model in the previous chapter) to have some knowledge about the internal behavior of the process itself.

## 7.5 Graph reduction

If the application context allows it, it can be possible to perform two kinds of reductions on an OPM graph using the completion rules defined in the model, obtaining in this way a more concise representation of the graph.

The first reduction can be applied if two processes are connected through an artifact – generated by one of the two processes and used by the other one – that is not used by other processes. If this artifact is considered not relevant for the application context, it can be eliminated inserting a *wasTriggeredBy* edge between the processes (see figure 6.2). It order not to loose relevant information, it is advisable not to perform this kind of transformation when the involved artifact is connected to another artifact through a *wasDerivedFrom* edge.

The second possible reduction is applicable when two artifacts are connected by a *wasDerivedFrom* edge and also linked through a process – using one of the two artifacts and generating the other one – that does not generate and does not use other artifacts. In this case, the intermediate process can be eliminated if it is not considered relevant for the application context. Moreover, if there is an agent linked only to the deleted process, it will be removed as well from the graph.

## 7.6 The method in practice: an example

To illustrate the described design approach, we consider a simplified version of the production procedure of the Indices of Consumer Prices provided by the Italian National Bureau of Census (Istituto Nazionale di Statistica, whose acronym is Istat). These indices are economic indicators that measure the change over time of the prices of consumer goods and services. In particular, Istat provides three different kinds of indices for consumer prices: an index for the whole population, called NIC; an index for employee families, called FOI; and the harmonized index of consumer prices, called IPCA or HICP used at the European Union level.


**Figure 7.4:** Istat example: first sketch

In order to compute these indices, Istat composes a "basket" of goods and services that reflects the representative consumption pattern of all types of households in the country. Data are collected in two ways: at central level, directly by Istat,

| Artifact | Generated by Processes | Used by Processes | Dependent on Artifacts | Sources for Artifacts | Flagged |
|---|---|---|---|---|---|
| NIC | NIC computation | Dissemination | Validated data | Report | |
| FOI | FOI computation | Dissemination | Validated data | Report | |
| IPCA | IPCA computation | Dissemination | Validated data | Report | |
| Prices | Collection of prices at local level, Collection of prices at centralized level | Data validation | | Validated data | X |
| Basket of products | Basket of products definition | Collection of prices at local level, Collection of prices at centralized level | | | |
| Product weight | Weight definition | NIC computation, FOI computation, IPCA computation | | | |
| Validated data | Data validation | NIC computation, FOI computation, IPCA computation | Prices | NIC, FOI, IPCA | |
| Report | Dissemination | | NIC, FOI, IPCA | | |

**Figure 7.5:** Istat example: the table for the artifacts

| Process | Generating Artifacts | Using Artifacts | Input/Output dependecies | Dependent on Processes | Controlled by Agents |
|---|---|---|---|---|---|
| Basket of products definition | Basket of products | | | | Consumer Prices Department (Istat) |
| Collection of prices at local level | Prices | Basket of products | | | Municipal office of statistics |
| Collection of prices at centralized level | Prices | Basket of products | | | Consumer Prices Department (Istat) |
| Data validation | Validated data | Prices | Validated data from Prices | | Consumer Prices Department (Istat) |
| Weight definition | Product weight | | | | Consumer Prices Department (Istat) |
| NIC computation | NIC | Validated data, Product weight | NIC from Validated data | | Consumer Prices Department (Istat) |
| FOI computation | FOI | Validated data, Product weight | FOI from Validated data | | Consumer Prices Department (Istat) |
| IPCA computation | IPCA | Validated data, Product weight | IPCA from Validated data | | Consumer Prices Department (Istat) |
| Dissemination | Report | NIC, FOI, IPCA | Report from NIC, FOI, IPCA | | Consumer Prices Department (Istat) |

**Figure 7.6:** Istat example: the table for the processes

| Artifact | Generated by Processes | Used by Processes | Dependent on Artifacts | Sources for Artifacts | Flagged |
|---|---|---|---|---|---|
| NIC | NIC computation | Dissemination | Validated data | Report | |
| FOI | FOI computation | Dissemination | Validated data | Report | |
| IPCA | IPCA computation | Dissemination | Validated data | Report | |
| Prices | Collection of prices at local level, Collection of prices at centralized level | Data validation | | Validated data | x |
| Basket of products | Basket of products definition | Collection of prices at local level, Collection of prices at centralized level | | | |
| Product weight | Weight definition | NIC computation, FOI computation, IPCA computation | | | |
| Validated data | Data validation | NIC computation, FOI computation, IPCA computation | Prices | NIC, FOI, IPCA | |
| Report | Dissemination | | NIC, FOI, IPCA | | |
| Prices | Data assembling | Data validation | | Validated data | |

**Figure 7.7:** Istat example: the modified table for the artifacts

| Process | Generating Artifacts | Using Artifacts | Input/Output dependecies | Dependent on Processes | Controlled by Agents |
|---|---|---|---|---|---|
| Basket of products definition | Basket of products | | | | Consumer Prices Department (Istat) |
| Collection of prices at local level | Prices | Basket of products | | | Municipal office of statistics |
| Collection of prices at centralized level | Prices | Basket of products | | | Consumer Prices Department (Istat) |
| Data validation | Validated data | Prices | Validated data from Prices | | Consumer Prices Department (Istat) |
| Weight definition | Product weight | | | | Consumer Prices Department (Istat) |
| NIC computation | NIC | Validated data, Product weight | NIC from Validated data | | Consumer Prices Department (Istat) |
| FOI computation | FOI | Validated data, Product weight | FOI from Validated data | | Consumer Prices Department (Istat) |
| IPCA computation | IPCA | Validated data, Product weight | IPCA from Validated data | | Consumer Prices Department (Istat) |
| Dissemination | Report | NIC, FOI, IPCA | Report from NIC, FOI, IPCA | | Consumer Prices Department (Istat) |
| Data assembling | Prices | | | Collection of prices at local level, Collection of prices at centralized level | Consumer Prices Department (Istat) |
| Collection of prices at local level | | Basket of products | | | Municipal office of statistics |
| Collection of prices at centralized level | | Basket of products | | | Consumer Prices Department (Istat) |

**Figure 7.8:** Istat example: the modified table for the processes

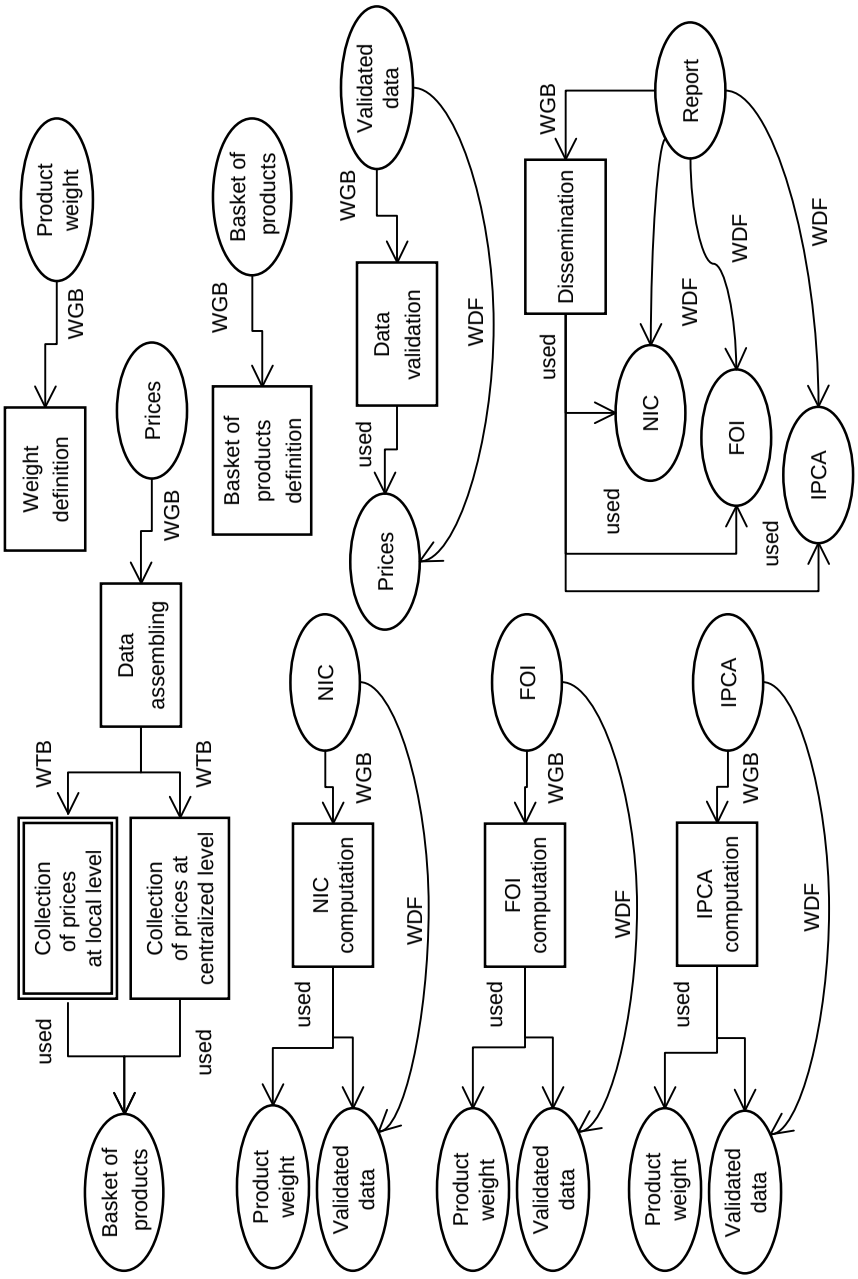



**Figure 7.9:** Istat example: the subgraphs

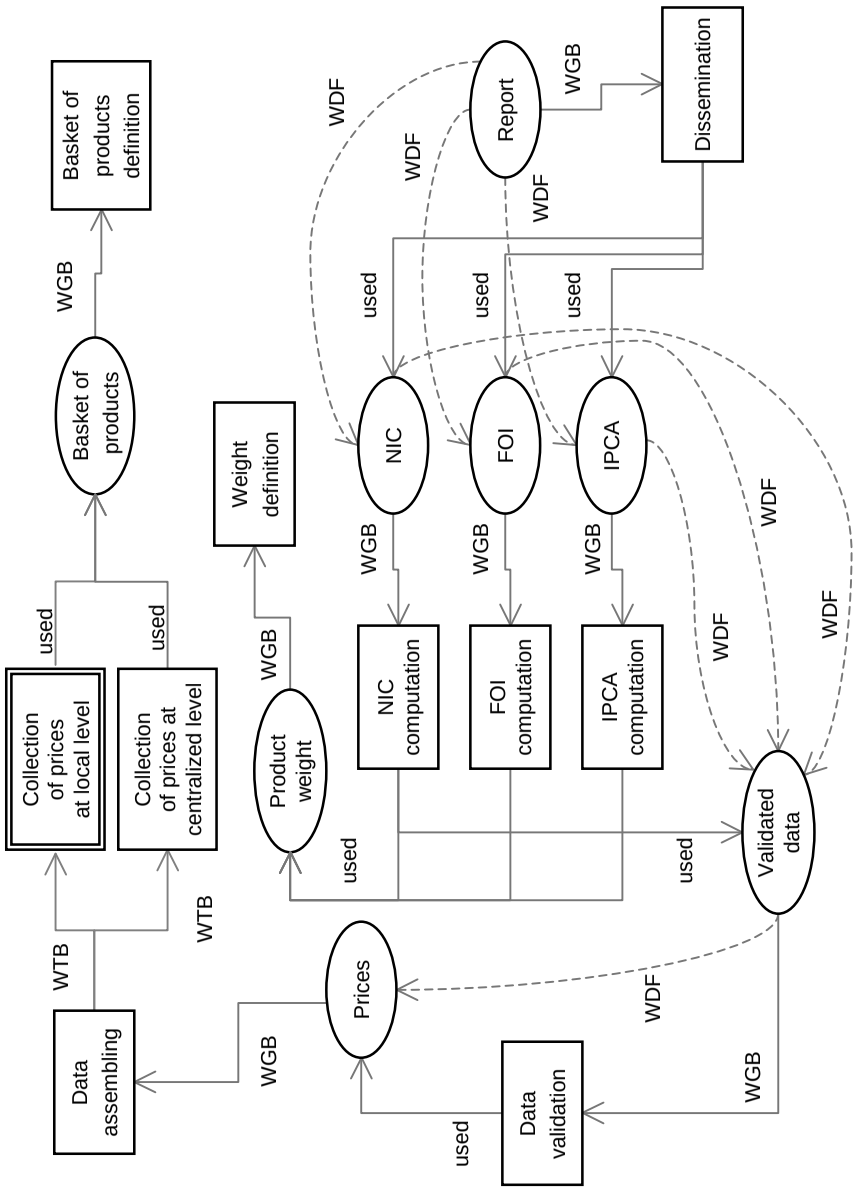



**Figure 7.10:** Istat example: the merged graph

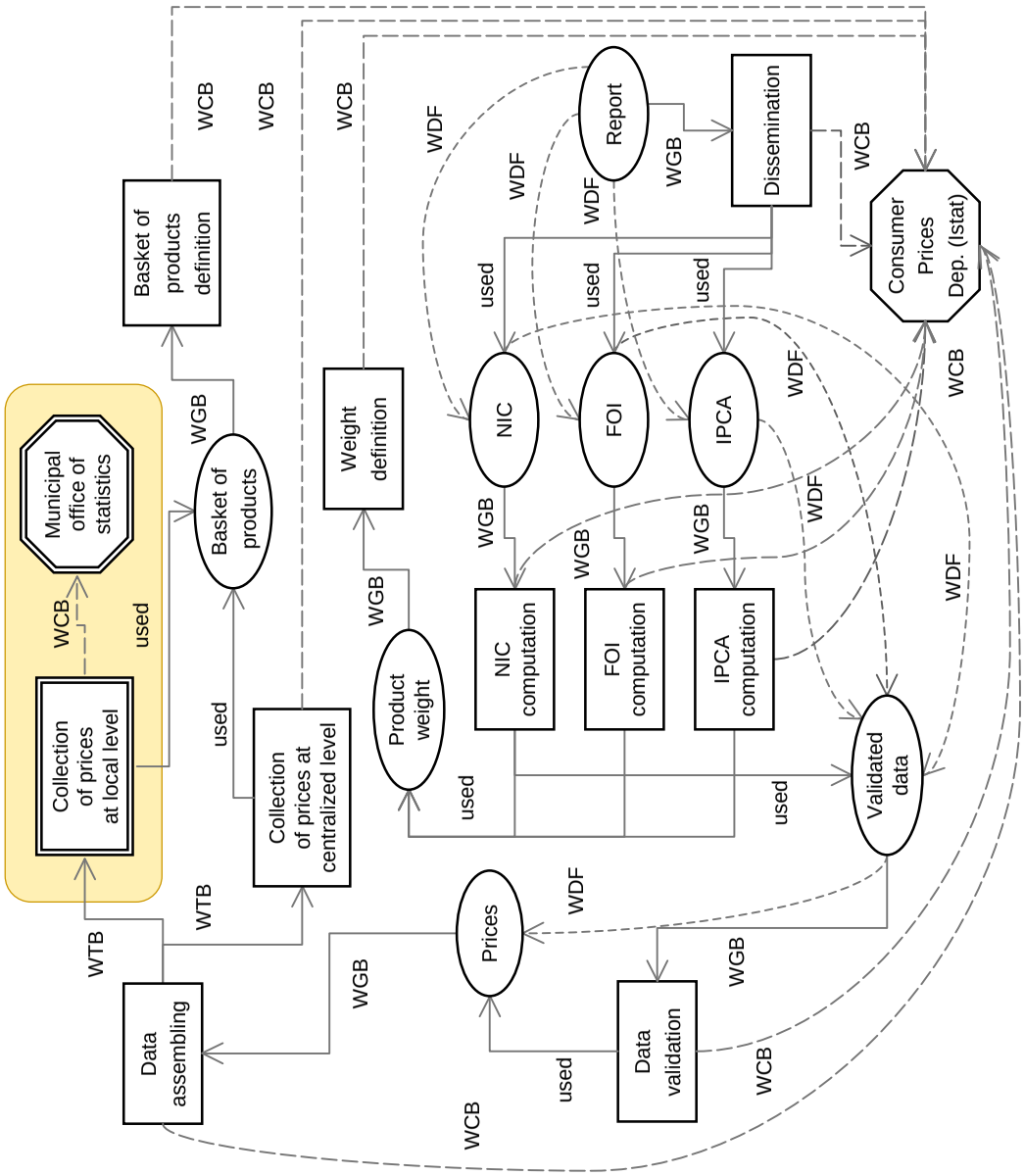



**Figure 7.11:** Istat example: the final graph

for goods and services having uniform prices on the whole national territory; at territorial level, by Municipal Offices of Statistics in the towns participating in the survey, prices are collected through questionnaires or palm PC and, later on, registered and checked. Not all the components in the basket have the same weight: food is purchased almost daily, fuel for a car perhaps weekly, and car insurance is paid once a year. So each item must be weighted to reflect its importance in a household budget (in the OPM graph the production processes for the nodes called *Basket of Products* and *Product Weight* are mentioned, but we do not provide the description of the processes producing these artifacts).

In this example, we refer to the legal graph definition proposed in the OPM version 1.1. The design steps of the OPM graph for the described example are shown in figures 7.4 – 7.11. Figure 7.4 shows a basic sketch, at a very high level of abstraction, of the leading artifacts. The tables listing respectively artifacts and processes are reported in figures 7.5 and 7.6. Figures 7.7 and 7.8 show the same two tables modified to solve the problem of having two processes generating the artifact called *Prices* (the shaded rows were deleted, while the new rows were added at the end of each table). Afterwards, figure 7.9 shows the subgraphs, figure 7.10 the merged graph, and figure 7.11 the complete graph with the agents (in the graphs built for the example the roles on the edges are not displayed and the names of the edges, except the one for the *used* relation, are shortened in the following way: WGB for *wasGeneratedBy*, WDF for *wasDerivedFrom*, WTB for *wasTriggeredBy*, and WCB for *wasControlledBy*; moreover, the *wasDerivedFrom* and *wasControlledBy* edges are drawn with dashed lines). In the final graph, shown in figure 7.11, the subgraph enclosed in the shaded box is an abstract subgraph generally representing the price collection activities performed by the Municipal Offices of Statistics. In the actual OPM graph, this abstract subgraph has to be replaced with the entities referring to each Municipal Office of Statistics and its data collection activity.

## 7.7 Is the designed graph legal?

It is a good practice to verify if the obtained OPM graph is legal, according to the chosen definition. If the graph contains one or more nodes that are not satisfying the definition of legal graph, then it has to be modified.

First of all, it is necessary to analyze if the cause is an incorrect representation of the relationships among the entities in the graph and correct them. Otherwise, if the relationships among the entities are correctly modeled, a different way of expressing the same meaning, but in accordance with the definition of legal OPM graph, has to be found.

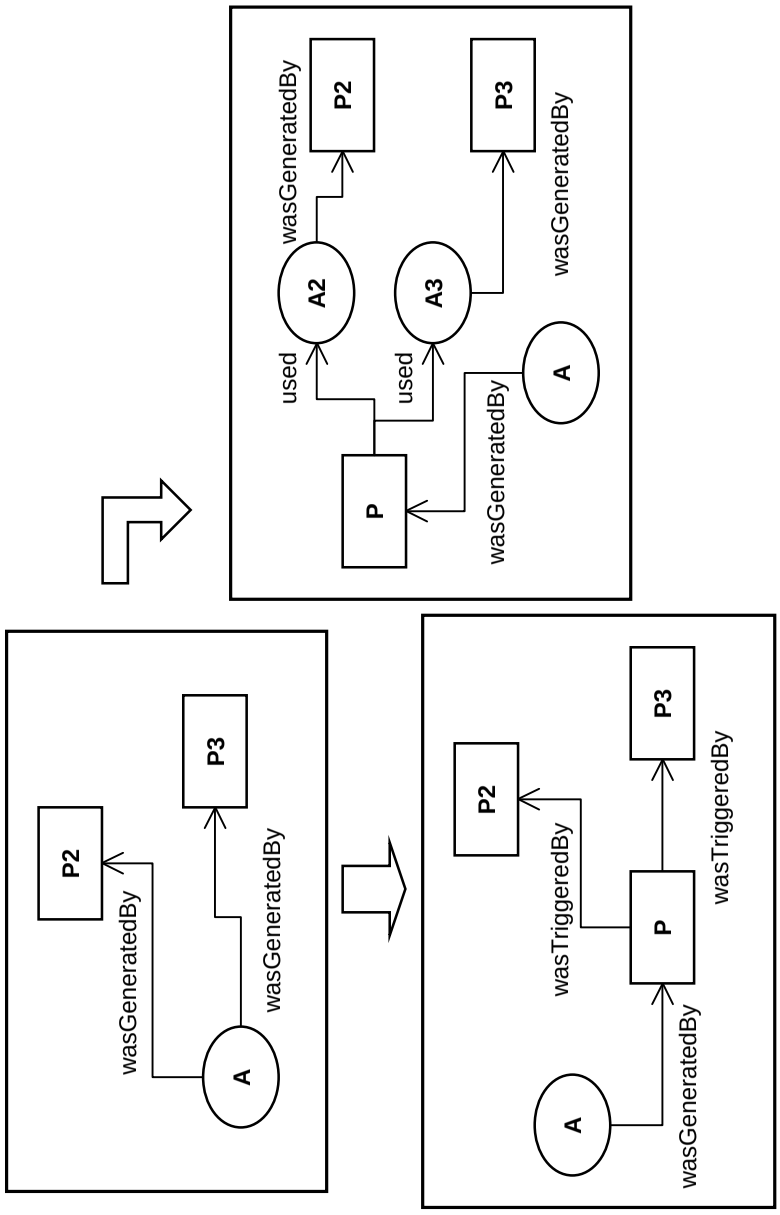



**Figure 7.12:** Transformation rule

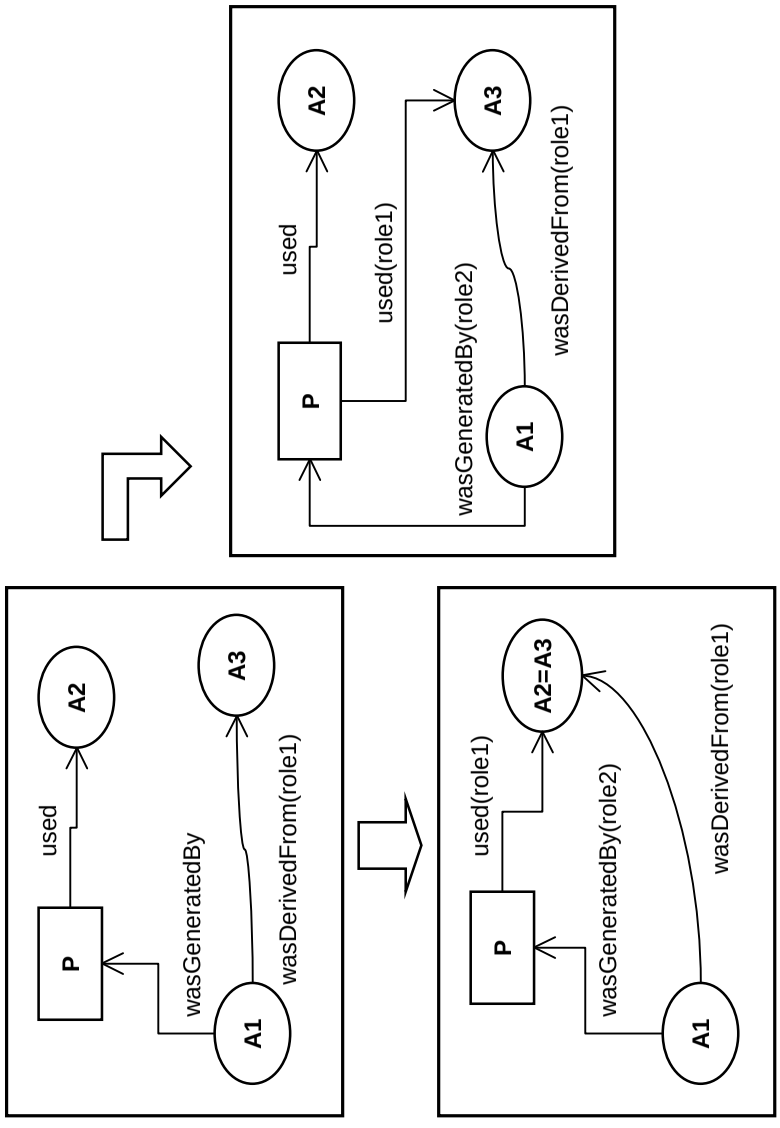



**Figure 7.13:** Transformation rule

A final verification of the constraints on the graph can be also executed after the serialization of the graph in the corresponding XML format, as explained later in the chapter.

### 7.7.1 Transformation hints

In order to reach the target to obtain a legal graph, we suggest some common transformations that can be used to modify a graph. The proposed transformation hints derive from the definitions of legal OPM graph and from the completion operations defined for the model.

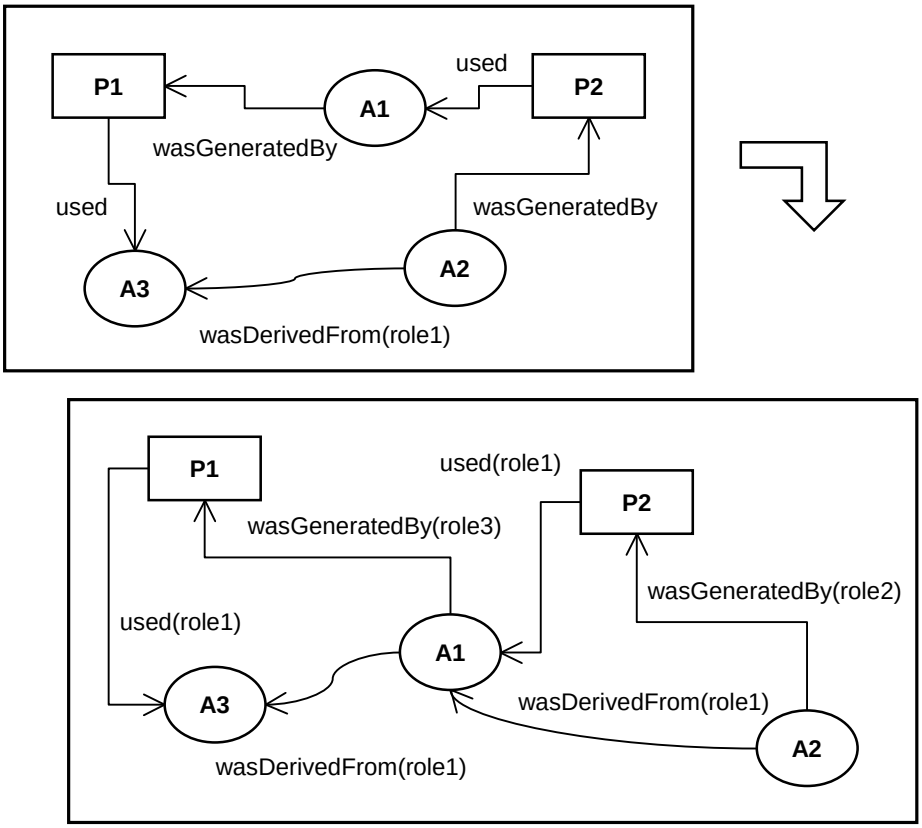



**Figure 7.14:** Transformation rule

The first graph shown in figure 7.12 is not legal with respect to the definition of legal OPM graph given in the version 1.1 of the model; in fact, in the graph there are two *wasGeneratedBy* edges for the same artifact A. This can happen when an artifact is generated using the data produced by more than one process. In this case, a possible solution (as it was already mentioned previously) is the insertion of

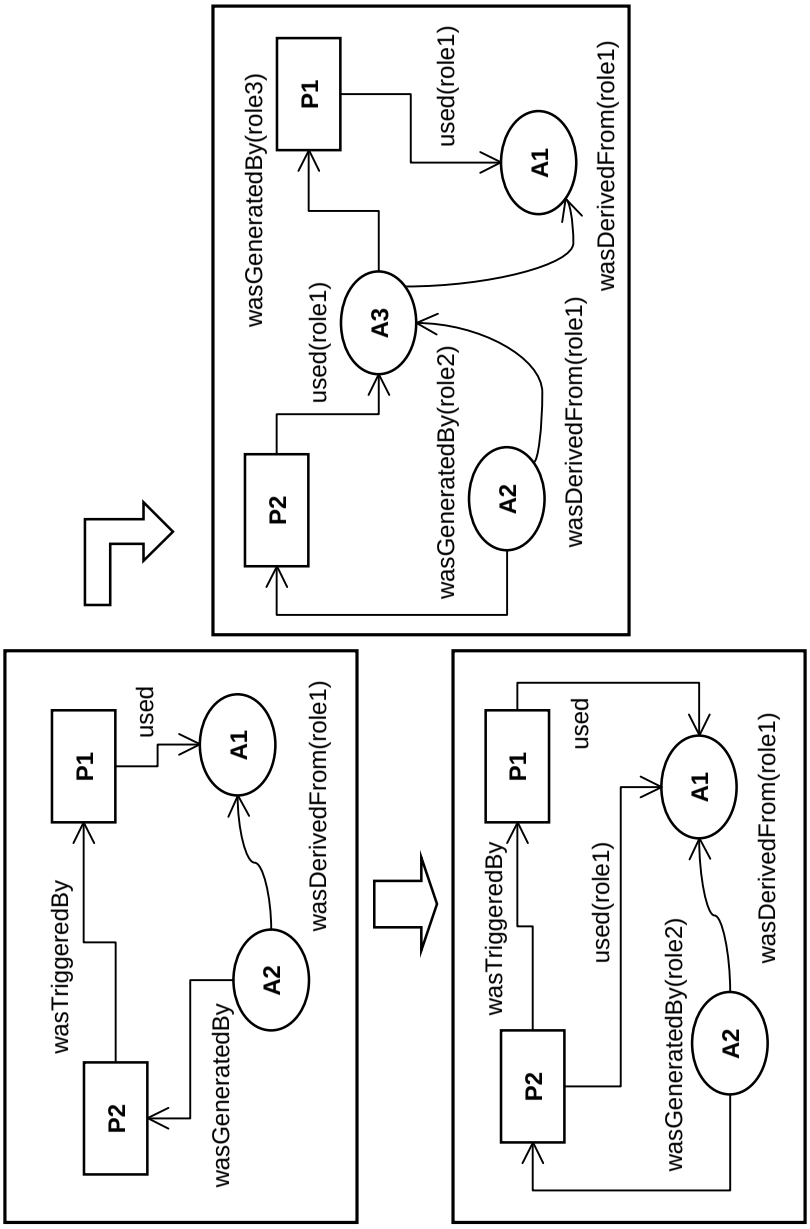



**Figure 7.15:** Transformation rule

an intermediate process merging the data produced by the original processes.

The other proposed transformations can apply to graphs that do not comply with the definition of legal graph introduced in [Kwasnikowska et al., 2010].

In figure 7.13, the shown graph is not legal because it does not satisfy the second condition contained in the definition. There are two possible solutions: or the artifacts A2 and A3 are in reality the same artifact, or a *used* edge connecting the process P with the artifact A3 has to be added. Figure 7.14 and 7.15 show other two examples of graphs that do not comply with the mentioned definition for the presence of a precise *wasDerivedFrom* edge without a use-generate-derive triangle; the corresponding transformations into legal forms are also shown.

### 7.7.2 Constraints on time information

The presence of timestamps on the edges of an OPM graph determines the need to execute a further check against the temporal constraints defined in the model. This check can be done on the graph comparing each timestamp with the timestamps of the contiguous edges, but it is easier to verify the temporal constraints after the serialization of the graph in the corresponding XML format, as explained later in the chapter.

## 7.8 Physical modeling

The target of this phase of the design is to transform the OPM conceptual graph into a physical format; in particular, we used the XML format.

The specification of the XML Schema for the version 1.1 of OPM can be found in [Groth and Moreau, 2010]. Not being aware of any published specifications for the XML structure of the OPM version described in [Kwasnikowska et al., 2010], we extended the XML Schema of the version 1.1.

The main changes are related to the *wasDerivedFrom* and the *wasGeneratedBy* edges. As shown in the following, the code for the *complexType* defining the *wasDerivedFrom* edge was modified adding the role element:

```
<xs:complexType name="WasDerivedFrom">
  <xs:sequence>
    <xs:element type="opmx:ArtifactRef" name="effect"/>
    <xs:element type="opmx:ArtifactRef" name="cause"/>
    <xs:element type="opmx:Role" name="role"
                                  minOccurs="0" maxOccurs="1"/>
    <xs:element type="opmx:AccountRef" name="account"
```

```
                          minOccurs="0" maxOccurs="unbounded"/>
    <xs:element type="opmx:OTime" name="time" minOccurs="0"/>
    <xs:element minOccurs="0" maxOccurs="unbounded"
                              ref="opmx:annotation"/>
  </xs:sequence>
  <xs:attribute type="xs:ID" name="id"/>
</xs:complexType>
```

Moreover, the code for the *complexType* that defines the *wasGeneratedBy* edge was modified changing the role element from mandatory to optional:

```
<xs:complexType name="WasGeneratedBy">
  <xs:sequence>
    <xs:element type="opmx:ArtifactRef" name="effect"/>
    <xs:element type="opmx:Role" name="role"
                                 minOccurs="0" maxOccurs="1"/>
    <xs:element type="opmx:ProcessRef" name="cause"/>
    <xs:element type="opmx:AccountRef" name="account"
                minOccurs="0" maxOccurs="unbounded"/>
    <xs:element type="opmx:OTime" name="time" minOccurs="0"/>
    <xs:element minOccurs="0" maxOccurs="unbounded"
                              ref="opmx:annotation"/>
  </xs:sequence>
  <xs:attribute type="xs:ID" name="id"/>
</xs:complexType>
```

We use a top-down approach to derive the XML tree structure from the graph. We start the top-down serialization creating an empty *opmGraph* tag. Accounts do not have a graphical representation, even if they can be distinguished by means of colors when it is necessary to draw them into a single graph; however, they need to be defined in the XML code. So we add an empty *accounts* tag and, for each account that is referred into the graph, an *account* tag with an appropriate identifier. After the accounts are defined, we add three empty tags called *artifacts*, *processes*, and *agents*, then under them we add every artifact, process, and agent. Afterwards, we insert an empty *dependencies* tag that will contain all the edges of the graph. In order to serialize the edges, for each process, we add the corresponding *used* and *wasTriggeredBy* edges; for each artifact, we add the corresponding *wasGeneratedBy* and *wasDerivedFrom* edges; for each agent, we add the corresponding *wasControlledBy* edges.

### 7.8.1 Physical model validation

After the serialization of an OPM graph in the XML format, it is possible to verify the correctness of the XML structure and also if the constraints related to the definition of legal OPM graph are satisfied.

We approached this task exploiting the potentialities of Schematron [van der Vlist, 2007], as we did for the validation of the XML codification of the data quality rules (see chapter 4). However, because of the central role played by XPath, cycles that could arise with the "was derived from" relations cannot be detected using native Schematron assertions. For this reason, the adopted solution consisted in using Schematron assertions with in addition an ad hoc XSLT function that checks the presence of cycles in the *wasDerivedFrom* edges. The developed XSLT function can be added to the Schematron schema and called into a Schematron assertion.

In the following, we list the more relevant checks among those performed through the Schematron schema and the corresponding code to express them:

– At most one *wasGeneratedBy* edge per artifact is allowed (OPM specification version 1.1):

```
<iso:rule context="dependencies">
  <iso:assert test="count(wasGeneratedBy) =
   count(wasGeneratedBy[not(effect/@ref =
   preceding-sibling::wasGeneratedBy/effect/@ref)])">
    At most one wasGeneratedBy edge per artifact is allowed
  </iso:assert>
</iso:rule>
```

– At most one precise *wasGeneratedBy* edge per artifact is allowed (legal OPM graph definition in [Kwasnikowska et al., 2010]):

```
<iso:rule context="dependencies">
  <iso:assert test="count(wasGeneratedBy[role]) =
   count(wasGeneratedBy[role and not(effect/@ref =
   preceding-sibling::wasGeneratedBy/effect/@ref)])">
    At most one precise wasGeneratedBy edge
    per artifact is allowed
  </iso:assert>
</iso:rule>
```

– A use-generate-derive triangle must exist for every precise *wasDerivedFrom* edge, with role $r$, connecting artifact A1 and artifact A2 (legal OPM graph definition in [Kwasnikowska et al., 2010]); that is, two more edges are necessary: a precise *wasGeneratedBy* edge connecting A1 to a process P and a precise *used* edge, with role $r$, connecting the same process P to A2.

```
<iso:rule context="wasDerivedFrom">
  <let name="a1ID" value="effect/@ref"/>
  <let name="a2ID" value="cause/@ref"/>
  <iso:assert test="role and
   ../wasGeneratedBy[role and effect/@ref=$a1ID]/cause/@ref =
   ../used[role and cause/@ref=$a2ID]/effect/@ref and
   ../used[cause/@ref=$a2ID]/role/@id=role/@id">
    For every precise wasDerivedFrom edge connecting
    artifact A1 and artifact A2, a wasGeneratedBy edge
    connecting A1 with a process and a used edge
    connecting the same process to A2 are necessary
  </iso:assert>
</iso:rule>
```

– A *wasTriggeredBy* edge can connect only distinct processes (i.e., if P1 → P2 then P1≠P2):

```
<iso:rule context="wasTriggeredBy">
  <iso:assert test="not(effect/@ref=cause/@ref)">
    The processes connected by a wasTriggeredBy edge
    have to be distinct
  </iso:assert>
</iso:rule>
```

– A *wasDerivedFrom* edge can connect only distinct artifacts (i.e., if A1 → A2 then A1≠A2):

```
<iso:rule context="wasDerivedFrom">
  <iso:assert test="not(effect/@ref=cause/@ref)">
    The artifacts connected by a wasDerivedFrom edge
    have to be distinct
  </iso:assert>
</iso:rule>
```

Other checks performed with Schematron implement the temporal constraints defined in the model.

The complete Schematron schema can be found in appendix B.

## 7.9 Comments and related works

Data provenance is an important topic for the evaluation of the trustworthiness of a dataset, but it is not yet enough considered as a standard component in the data management systems both in industrial and in scientific contexts.

OPM, which is a model to represent provenance information, has been proposed as an interchange format among different systems.

In this work, we presented an approach that allows the design of OPM graphs satisfying the two definitions of legal graph provided in literature. The proposed approach is composed of two phases: in the conceptual phase, concepts and their relationships are identified and mapped into OPM elements; while the physical phase refers to the serialization of the OPM graph into the corresponding XML structure with the validation of the generated XML document.

Although OPM has been used in connection with different provenance systems, we are not aware of any methods or methodologies for the design of OPM graphs. In a different context of the data provenance field, however, a software engineering methodology called Provenance Incorporating Methodology (PrIMe) has been proposed [Miles et al., 2011].

PrIMe is a guided approach for making applications provenance-aware. Its target is the identification of the changes necessary to enable an application to interact with a provenance middleware layer. PrIMe is based on three different phases: (1) the identification of provenance use cases and the pieces of information that will be used to answer them, (2) the decomposition of the application into a set of actors and their interactions, and (3) the implementation of a set of adaptations to the application in order to ensure that the required information items are available for documentation.

CHAPTER 8

# Storing and Querying OPM Graphs

OPM graphs can be used to accompany datasets in order to show their provenance information to final users. In particular, it can be useful for data that are published on the World Wide Web. However, to be effective for a real-world application, a provenance tool should provide facilities to query provenance information and data items. To provide functionalities of this kind, it may be an advantage to have the provenance information stored with the data themselves. To achieve this target, when using an OPM graph to model provenance information, it is possible to store in a repository the graph and connect it to the archive of the original data, having therefore a straightforward way to perform queries on both the graph and the data.

In this chapter, we describe the solution adopted to store an OPM graph into a relational database and the approach developed to query the information contained in the OPM graph.

## 8.1 Storing OPM graphs

To store OPM graphs in a relational database, firstly we designed an Entity Relationship diagram [Elmasri and Navathe, 2000] for the OPM model and translated it into a relational schema; the second step was the definition of the connection with the data. In figure 8.1 an Entity Relationship diagram for the OPM model is shown (we refer to the OPM specification version 1.1). From the Entity Relationship diagram the following relational schema was derived:

```
Artifact(artifactId, value)
Process(processId, value)
Agent(agentId, value)
WasDerivedFrom(wasDerivedFromId, derivedArtifactId, sourceArtifactId)
```

```
Unique key [derivedArtifactId, sourceArtifactId]
WasGeneratedBy(wasGeneratedById, artifactId, processId, role)
Unique key [artifactId, processId, role]
Used(usedId, processId, artifactId, role)
Unique key [processId, artifactId, role]
WasControlledBy(wasControlledById, processId, agentId, role)
Unique key [processId, agentId, role]
WasTriggeredBy(wasTriggeredById, triggeredProcessId, sourceProcessId)
Unique key [triggeredProcessId, sourceProcessId]
ArtifactAnnotation(artifactId, property, value)
AgentAnnotation(agentId, property, value)
ProcessAnnotation(processId, property, value)
WasDerivedFromAnnotation(wasDerivedFromId, property, value)
WasGeneratedByAnnotation(wasGeneratedById, property, value)
UsedAnnotation(usedId, property, value)
WasControlledByAnnotation(wasControlledById, property, value)
WasTriggeredByAnnotation(wasTriggeredById, property, value)
ArtifactAccount(artifactId, accountName)
AgentAccount(agentId, accountName)
ProcessAccount(processId, accountName)
WasDerivedFromAccount(wasDerivedFromId, accountName)
WasGeneratedByAccount(wasGeneratedById, accountName)
UsedAccount(usedId, accountName)
WasControlledByAccount(wasControlledById, accountName)
WasTriggeredByAccount(wasTriggeredById, accountName)
WasDerivedFromTime(wasDerivedFromId, time)
WasGeneratedByTime(wasGeneratedById, time)
UsedTime(usedId, time)
WasControlledByTime(wasControlledById, startTime, endTime)
WasTriggeredByTime(wasTriggeredById, time)
```

The OPM graph developed for the case study was serialized in the XML format (the XML Schema specification for OPM version 1.1 can be found in [Groth and Moreau, 2010]) and then imported in the PostgreSQL database.

### 8.1.1 Views for multi-step relations

As already introduced in chapter 6, the multi-step *wasDerivedFrom* edge and three secondary multi-step edges were defined for OPM in [Moreau et al., 2011] to allow

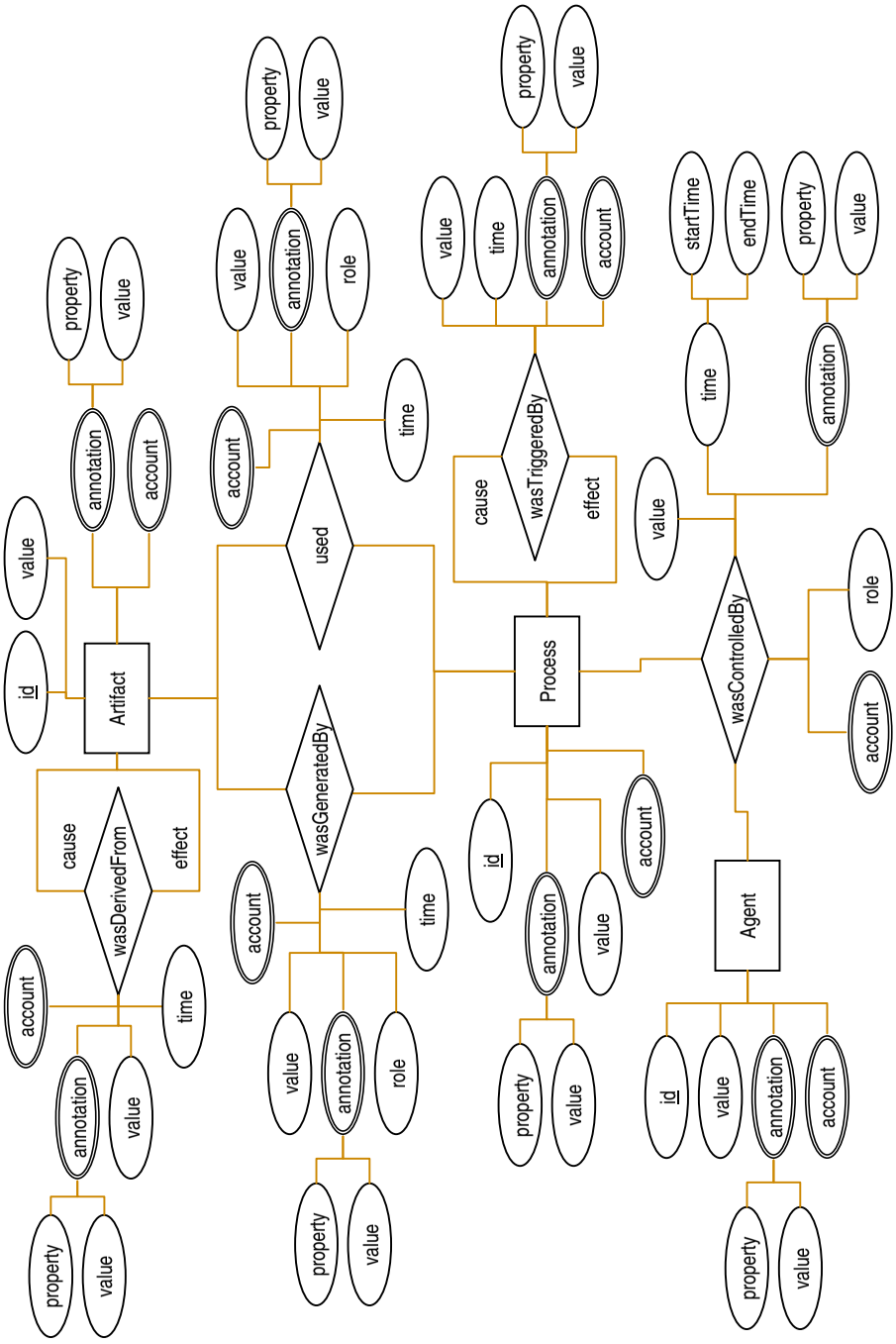



**Figure 8.1:** An Entity Relationship diagram for OPM graphs

the expression of queries and inferences in OPM graphs.

We implemented the four multi-step edges defined in the model ($A1 \rightarrow *A2$, $P \rightarrow *A$, $A \rightarrow *P$, and $P2 \rightarrow *P1$) and the new one proposed in this work ($A \rightarrow *Ag$, see chapter 6) in the PostgreSQL database through `views`. In particular, the multi-step *wasDerivedFrom* $A1 \rightarrow *A2$ edge was defined using the SQL `WITH RECURSIVE` clause which implements the Common Table Expression (CTE) defined in the SQL:1999 standard and available in PostgreSQL. In the database systems that do not support the CTE, the same result can be achieved implementing the necessary routines to extract the desired links among artifacts, for example, using stored procedures.

The defined `views` allow an easy approach in querying multi-step inferences on OPM graphs.

**Multi-step *wasDerivedFrom* A1→*A2**

```
CREATE OR REPLACE VIEW MultiStepWasDerivedFrom AS
WITH RECURSIVE tempWasDerivedFromView(derivedArtifactId,
                                      sourceArtifactId) AS (
  SELECT derivedArtifactId, sourceArtifactId, accountName
  FROM WasDerivedFrom, WasDerivedFromAccount
  WHERE WasDerivedFrom.wasDerivedFromId =
        WasDerivedFromAccount.wasDerivedFromId
  UNION
  SELECT T.derivedArtifactId, WasDerivedFrom.sourceArtifactId,
         T.accountName
  FROM WasDerivedFrom, WasDerivedFromAccount,
       tempWasDerivedFromView T
  WHERE WasDerivedFrom.derivedArtifactId = T.sourceArtifactId
  AND WasDerivedFrom.wasDerivedFromId =
      WasDerivedFromAccount.wasDerivedFromId
  AND T.accountName = WasDerivedFromAccount.accountName
)
SELECT derivedArtifactId, sourceArtifactId, accountName
FROM tempWasDerivedFromView
ORDER BY sourceArtifactId, derivedArtifactId, accountName
```

**Multi-step *used* P→*A**

```
CREATE OR REPLACE VIEW MultiStepUsed AS
  SELECT processId, artifactId, accountName
```

```
  FROM Used, UsedAccount
  WHERE Used.usedId = UsedAccount.usedId
  UNION
  SELECT Used.processId, MultiStepWasDerivedFrom.sourceArtifactId,
         UsedAccount.accountName
  FROM Used, UsedAccount, MultiStepWasDerivedFrom
  WHERE Used.artifactId = MultiStepWasDerivedFrom.derivedArtifactId
  AND Used.usedId = UsedAccount.usedId
  AND MultiStepWasDerivedFrom.accountName = UsedAccount.accountName
```

**Multi-step *wasGeneratedBy* A→*P**

```
CREATE OR REPLACE VIEW MultiStepWasGeneratedBy AS
  SELECT artifactId, processId, accountName
  FROM WasGeneratedBy, WasGeneratedByAccount
  WHERE WasGeneratedBy.wasGeneratedById =
        WasGeneratedByAccount.wasGeneratedById
  UNION
  SELECT MultiStepWasDerivedFrom.derivedArtifactId,
         WasGeneratedBy.processId, WasGeneratedByAccount.accountName
  FROM WasGeneratedBy, WasGeneratedByAccount, MultiStepWasDerivedFrom
  WHERE WasGeneratedBy.artifactId =
        MultiStepWasDerivedFrom.sourceArtifactId
  AND WasGeneratedBy.wasGeneratedById =
      WasGeneratedByAccount.wasGeneratedById
  AND MultiStepWasDerivedFrom.accountName =
      WasGeneratedByAccount.accountName
```

**Multi-step *wasTriggeredBy* P2→*P1**

```
CREATE OR REPLACE VIEW MultiStepWasTriggeredBy AS
  SELECT triggeredProcessId, sourceProcessId, accountName
  FROM WasTriggeredBy, WasTriggeredByAccount
  WHERE WasTriggeredBy.wasTriggeredById =
        WasTriggeredByAccount.wasTriggeredById
  UNION
  SELECT Used.processId, WasGeneratedBy.processId,
         UsedAccount.accountName
  FROM Used, WasGeneratedBy, UsedAccount, WasGeneratedByAccount
```

```
  WHERE WasGeneratedBy.artifactId = Used.artifactId
  AND WasGeneratedBy.wasGeneratedById =
      WasGeneratedByAccount.wasGeneratedById
  AND Used.usedId = UsedAccount.usedId
  UNION
  SELECT Used.processId, WasGeneratedBy.processId,
         UsedAccount.accountName
  FROM Used, WasGeneratedBy, UsedAccount,
       WasGeneratedByAccount, MultiStepWasDerivedFrom
  WHERE WasGeneratedBy.artifactId =
        MultiStepWasDerivedFrom.sourceArtifactId
  AND Used.artifactId =
      MultiStepWasDerivedFrom.derivedArtifactId
  AND WasGeneratedBy.wasGeneratedById =
      WasGeneratedByAccount.wasGeneratedById
  AND Used.usedId = UsedAccount.usedId
  AND MultiStepWasDerivedFrom.accountName =
      WasGeneratedByAccount.accountName
  AND MultiStepWasDerivedFrom.accountName = UsedAccount.accountName
```

### Multi-step artifact-agent edge A→*Ag

The following view implements the multi-step edge between an artifact and an agent as defined in the second paragraph of chapter 6.

```
CREATE OR REPLACE VIEW MultiStepArtifactWasControlledBy AS
  SELECT artifactId, agentId, accountName
  FROM WasControlledBy, WasGeneratedBy,
       WasControlledByAccount, WasGeneratedByAccount
  WHERE WasControlledBy.processId = WasGeneratedBy.processId
  AND WasControlledBy.wasControlledById =
      WasControlledByAccount.wasControlledById
  AND WasGeneratedBy.wasGeneratedById =
      WasGeneratedByAccount.wasGeneratedById
  AND WasControlledByAccount.accountName =
      WasGeneratedByAccount.accountName
  UNION
  SELECT MultiStepWasDerivedFrom.derivedArtifactId,
         WasControlledBy.agentId, WasControlledByAccount.accountName
```

```
FROM WasControlledBy, WasGeneratedBy, WasControlledByAccount
     WasGeneratedByAccount, MultiStepWasDerivedFrom
WHERE WasGeneratedBy.artifactId =
      MultiStepWasDerivedFrom.sourceArtifactId
AND WasControlledBy.processId = WasGeneratedBy.processId
AND WasControlledBy.wasControlledById =
    WasControlledByAccount.wasControlledById
AND WasGeneratedBy.wasGeneratedById =
    WasGeneratedByAccount.wasGeneratedById
AND WasControlledByAccount.accountName =
    WasGeneratedByAccount.accountName
AND MultiStepWasDerivedFrom.accountName =
    WasGeneratedByAccount.accountName
```

### 8.1.2 OPM graph validation

Having an OPM graph stored in a relational database, it is easy to check if the graph itself is valid with respect to the definitions provided for the model and also if the timestamps expressed on the edges are consistent. In the developed validation step both the definitions of legal OPM graph were considered.

For the definition given in the model specification version 1.1 [Moreau et al., 2011], for each artifact at most one *wasGeneratedBy* edge is required. To identify the records that do not comply with this condition, it is possible to verify if in the `WasGeneratedBy` table there is more than one record with the same `artifactId` and `accountName`, for example, through the following SQL statement:

```
SELECT artifactId, accountName, COUNT(*)
FROM WasGeneratedBy, WasGeneratedByAccount
WHERE WasGeneratedBy.wasGeneratedById =
      WasGeneratedByAccount.wasGeneratedById
GROUP BY artifactId, accountName HAVING COUNT(*)>1
```

For the definition used in [Kwasnikowska et al., 2010], the first condition to be verified is that an artifact can have at most one precise *wasGeneratedBy* edge. This can be done considering the role in the `WasGeneratedBy` table:

```
SELECT artifactId, accountName, COUNT(*)
FROM WasGeneratedBy, WasGeneratedByAccount
WHERE WasGeneratedBy.wasGeneratedById =
      WasGeneratedByAccount.wasGeneratedById
```

```
AND WasGeneratedBy.role IS NOT NULL
GROUP BY artifactId, accountName HAVING COUNT(*)>1
```

The second condition to be verified is the existence of a use-generate-derive triangle for each precise *wasDerivedFrom* edge. Namely, for a precise *wasDerivedFrom* edge connecting artifact A1 and artifact A2, it is required to have a process linked to A1 through a precise *wasGeneratedBy* edge and to A2 through a precise *used* edge; moreover, the *used* edge has to be characterized by the same role present in the *wasDerivedFrom* edge. In order to deal with this constraint, it was necessary to modify the table `WasDerivedFrom` adding the role for the "was derived from" relation (for this kind of edge the role was not considered in the OPM version 1.1).

```
WasDerivedFromBis(wasDerivedFromId, derivedArtifactId,
                  sourceArtifactId, role)
Unique key [derivedArtifactId, sourceArtifactId, role]
```

Using this new table, the following SQL statement selects all the artifacts that do not comply with the given constraint.

```
SELECT derivedArtifactId FROM WasDerivedFromBis
WHERE derivedArtifactId NOT IN ( SELECT derivedArtifactId
  FROM WasDerivedFromBis, WasGeneratedBy, Used,
       WasDerivedFromAccount, WasGeneratedByAccount, UsedAccount
  WHERE WasGeneratedBy.processId = Used.processId
  AND WasDerivedFromBis.derivedArtifactId = WasGeneratedBy.artifactId
  AND WasDerivedFromBis.sourceArtifactId = Used.artifactId
  AND WasGeneratedBy.wasGeneratedById =
      WasGeneratedByAccount.wasGeneratedById
  AND WasDerivedFromBis.wasDerivedFromId =
      WasDerivedFromAccount.wasDerivedFromId
  AND Used.usedId = UsedAccount.usedId
  AND WasGeneratedByAccount.accountName = UsedAccount.accountName
  AND WasDerivedFromAccount.accountName = UsedAccount.accountName
  AND WasGeneratedBy.role IS NOT NULL
  AND WasDerivedFromBis.role IS NOT NULL
  AND Used.role IS NOT NULL AND WasDerivedFromBis.role = Used.role )
```

A check valid for both the given definitions is the one related to the presence of cycles in a graph: an OPM valid view – for a single account – does not contain "was derived from" cycles. This condition can be easily verified by means of the view implementing the multi-step *wasDerivedFrom* edge. Other relevant checks implement

the OPM temporal constraints and are used when timestamps are specified on the graph edges.

### 8.1.3 Mapping datasets to OPM graphs

OPM graphs stored in a relational database can be queried for artifacts provenance information and for relations among artifacts, processes, and agents. However, to be able to answer also questions about provenance information related to the actual data, it is necessary to connect provenance information with the real datasets.

In the context of the case study used in this work, the data received from Member States are stored in a PostgreSQL database. The data upload procedure registers in the database, together with the data, also the information about the sender and the reception date. The connection with the OPM graph is performed through tables linking the rows in the database tables containing the data received from Member States with the corresponding artifacts in the graph.

## 8.2 Querying OPM graphs

In order to access the information stored in an OPM graph, a set of queries were made available to the users. The queries can be classified as: queries about entities depending on other entities and queries about entities influencing other entities.

The query language grammar, described in the Extended Backus-Naur Form (EBNF) notation, is defined as follows:

```
<query specification>
      ::= Select <entity> <relation name> <entity> <where clause>
<entity>          ::= artifact | process | agent
<relation>        ::= dependsOn | hasEffectOn
<where clause>    ::= where <attribute> <operator> string
<attribute>       ::= id | value
<operator>        ::=  =
```

The available queries are listed in the following with their translation into SQL statements. It is made the assumption that the queries refer to a legal OPM account view; thus, accounts are not considered in the exemplified SQL code.

– `Select Artifact dependsOn Process where value = 'value'`

  The query selects the artifacts that satisfy the multi-step “was generated by” relation with the process indicated in the second part of the statement.

```
SELECT Artifact.value
FROM MultiStepWasGeneratedBy, Process, Artifact
WHERE MultiStepWasGeneratedBy.artifactId = Artifact.artifactId
AND MultiStepWasGeneratedBy.processId = Process.processId
AND Process.value = 'value'
```

– `Select Artifact dependsOn Artifact where value = 'value'`

The query selects the artifacts that satisfy the multi-step "was derived from" relation with the artifact indicated in the second part of the statement.

```
SELECT Artifact.value FROM Artifact
WHERE Artifact.artifactId in (
  SELECT MultiStepWasDerivedFrom.derivedArtifactId
  FROM MultiStepWasDerivedFrom, Artifact
  WHERE MultiStepWasDerivedFrom.sourceArtifactId =
        Artifact.artifactId AND Artifact.value = 'value' )
```

– `Select Artifact dependsOn Agent where value = 'value'`

The query selects the artifacts that indirectly depend on the agent indicated in the second part of the statement. Two SQL statement are exemplified; the first one uses the view for the proposed multi-step edge between an artifact and an agent.

```
SELECT Artifact.value
FROM Agent, Artifact, MultiStepArtifactWasControlledBy
WHERE MultiStepArtifactWasControlledBy.agentId = Agent.agentId
AND MultiStepArtifactWasControlledBy.artifactId =
    Artifact.artifactId AND Agent.value = 'value'
```

```
SELECT Artifact.value
FROM Agent, Artifact, WasControlledBy, MultiStepWasGeneratedBy
WHERE WasControlledBy.agentId = Agent.agentId
AND MultiStepWasGeneratedBy.artifactId = Artifact.artifactId
AND MultiStepWasGeneratedBy.processId =
    WasControlledBy.processId AND Agent.value = 'value'
```

– `Select Process dependsOn Artifact where value = 'value'`

The query selects the processes that satisfy the multi-step "used" relation with the artifact indicated in the second part of the statement.

```
SELECT Process.value
FROM MultiStepUsed, Process, Artifact
WHERE MultiStepUsed.artifactId = Artifact.artifactId
AND MultiStepUsed.processId = Process.processId
AND Artifact.value = 'value'
```

– `Select Process dependsOn Process where value = 'value'`

The query selects the processes that satisfy the multi-step "was triggered by" relation with the process indicated in the second part of the statement.

```
SELECT Process.value FROM Process
WHERE Process.processId in (
  SELECT MultiStepWasTriggeredBy.triggeredProcessId
  FROM MultiStepWasTriggeredBy, Process
  WHERE MultiStepWasTriggeredBy.sourceProcessId =
        Process.processId AND Process.value = 'value' )
```

– `Select Process dependsOn Agent where value = 'value'`

The query selects the processes that satisfy the "was controlled by" relation with the agent indicated in the second part of the statement.

```
SELECT Process.value
FROM WasControlledBy, Agent, Process
WHERE WasControlledBy.processId = Process.processId
AND WasControlledBy.agentId = Agent.agentId
AND Agent.value = 'value'
```

– `Select Artifact hasEffectOn Process where value = 'value'`

The query selects the artifacts that satisfy the inverse of the multi-step "used" relation with the process indicated in the second part of the statement.

```
SELECT Artifact.value
FROM MultiStepUsed, Process, Artifact
WHERE MultiStepUsed.artifactId = Artifact.artifactId
AND MultiStepUsed.processId = Process.processId
AND Process.value = 'value'
```

– `Select Artifact hasEffectOn Artifact where value = 'value'`

The query selects the artifacts that satisfy the multi-step "was derived from" relation with the artifact indicated in the second part of the statement.

```
SELECT Artifact.value FROM Artifact
WHERE Artifact.artifactId in (
  SELECT MultiStepWasDerivedFrom.sourceArtifactId
  FROM MultiStepWasDerivedFrom, Artifact
  WHERE MultiStepWasDerivedFrom.derivedArtifactId =
   Artifact.artifactId AND Artifact.value = 'value' )
```

– `Select Process hasEffectOn Artifact where value = 'value'`

The query selects the processes that satisfy the inverse of the multi-step "was generated by" relation with the artifact indicated in the second part of the statement.

```
SELECT Process.value
FROM MultiStepWasGeneratedBy, Process, Artifact
WHERE MultiStepWasGeneratedBy.artifactId = Artifact.artifactId
AND MultiStepWasGeneratedBy.processId = Process.processId
AND Artifact.value = 'value'
```

– `Select Process hasEffectOn Process where value = 'value'`

The query selects the processes that satisfy the inverse of the multi-step "was triggered by" relation with the process indicated in the second part of the statement.

```
SELECT Process.value FROM Process
WHERE Process.processId in (
  SELECT MultiStepWasTriggeredBy.sourceProcessId
  FROM MultiStepWasTriggeredBy, Process
  WHERE MultiStepWasTriggeredBy.triggeredProcessId =
        Process.processId AND Process.value = 'value' )
```

– `Select Agent hasEffectOn Process where value = 'value'`

The query selects the agents that satisfy the inverse of the "was controlled by" relation with the process indicated in the second part of the statement.

```
SELECT Agent.value
FROM WasControlledBy, Agent, Process
WHERE WasControlledBy.processId = Process.processId
AND WasControlledBy.agentId = Agent.agentId
AND Process.value = 'value'
```

– `Select Agent hasEffectOn Artifact where value = 'value'`

The query selects the agents that influence the artifact indicated in the second part of the statement. Two SQL statements are exemplified; in the first one the *MultiStepArtifactWasControlledBy* view is used.

```
SELECT Agent.value
FROM Agent, Artifact, MultiStepArtifactWasControlledBy
WHERE MultiStepArtifactWasControlledBy.agentId = Agent.agentId
AND MultiStepArtifactWasControlledBy.artifactId =
    Artifact.artifactId AND Artifact.value = 'value'

SELECT Agent.value
FROM Agent, Artifact, WasControlledBy, MultiStepWasGeneratedBy
WHERE WasControlledBy.agentId = Agent.agentId
AND MultiStepWasGeneratedBy.artifactId = Artifact.artifactId
AND MultiStepWasGeneratedBy.processId =
    WasControlledBy.processId AND Artifact.value = 'value'
```

## 8.3 Related works

In the context of the Third Provenance Challenge[1] and after it, some of the existing workflow systems managing data provenance (such as Karma, Kepler, and Taverna) were extended to support OPM or to map to OPM the model used in the system. Furthermore, new works have been carried out in order to deal with OPM.

OPMProv [Lim et al., 2011] is a workflow provenance system based on a relational database and compliant with OPM; in the OPMProv system, OPM graphs are stored in a relational database with an approach similar to the one used in our work.

Zhao et al. [2011] use OPM as a global provenance model to describe provenance information in a framework aiming at integrating heterogeneous data formats used by proprietary software systems; in the referenced work, the authors describe the implemented approach of reconstructing and querying the provenance graph from distributed provenance repositories using a limited set of operations exposed by the repository services.

In [Ding et al., 2010] the Semantic Web technologies have been exploited to implement OPM reasoning: an OWL ontology has been proposed to capture the concepts of OPM and valid inferences on the model itself.

[1] http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChallenge

# Conclusions

The importance of data quality is generally understood and recognized, but it is not always clear which is the best method to cope with the problem; moreover, the available tools do not solve all the issues, leading often to the need to develop ad hoc solutions.

In this work, through the application of data quality rules and with the adoption of the data provenance concept, we tried to pursue the ambitious target of recognizing if the quality level of a dataset is fit for the intended use of the data.

To deal with inconsistencies among data and evaluate the quality level of a dataset, we proposed an approach based on rules expressed using the XML markup language, and we developed a tool to evaluate these rules on a dataset. As data quality rules, we used order dependencies and existence constraints as well as some of the data constraints and dependencies already proposed in the data quality field.

The proposed approach allows a user to define different types of rules in the same environment in a straightforward way. The developed tool manages a predefined set of rule types, but it can be easily extended in order to work with other types of rules. In order to express data quality rules in the tool, a user needs only to know the database schema structure of its dataset and the types of rules managed by the tool itself; he is not required to develop any software. The direct use of XML can also be avoided through the templates developed to facilitate the interaction with the tool.

Furthermore, we explored the field of discovering data quality rules from datasets. New data dependencies, which can be used as data quality rules, have been recently proposed in literature. For some of these dependencies there are not yet well-established methods or available software tools to find them in a dataset; for this reason, we implemented some algorithms for discovering the data dependencies used in the tool.

The second main concept on which we based our work is the provenance of data. In spite of the rich literature of the last years, data provenance is not yet a concept normally used in data management activities. The existing systems in which the

provenance of data is considered and traced are built for particular purposes and limited contexts. Moreover, the interoperability among different systems is poor. In this context, the proposal of the Open Provenance Model (OPM) is a positive step, as the W3C proposal of the PROV-DM model is for the World Wide Web, but it is far from enough.

In the case study context, to start to experiment with the data provenance concept, we proposed the adoption of the OPM model in order to exchange data provenance information among the bodies involved in the collection and use of the data. A query language for querying OPM graphs stored in a relational database was implemented; in addition, we provided an approach to design OPM graphs together with a validation tool for the XML format of the graphs themselves. The proposed approach is composed of a conceptual phase and a physical phase. In the conceptual phase, concepts and their relationships are identified and mapped into OPM elements; moreover, graph transformation hints are provided to be used in order to obtain a legal OPM graph. Finally, the physical phase refers to the serialization of the OPM graph into the corresponding XML structure with the validation of the generated XML document.

# Appendix A

## Data Quality rules: XML structure

This appendix contains the Document Type Definition and the XML Schema for the XML structure used to specify the data quality rules as described in chapter 4.

### Document Type Definition

```
<!DOCTYPE rules [
<!ELEMENT rules (rule_definition+)>
<!ELEMENT rule_definition (table_name+,
  (rule_cr|rule_fd|rule_od|rule_dd|rule_ec|rule_cc))>
<!ELEMENT rule_cr (if, then)>
<!ELEMENT rule_fd (lhs, rhs, when?)>
<!ELEMENT rule_od (lhs, rhs, when?, partition_on?)>
<!ELEMENT rule_dd (lhs_dd, rhs_dd, when?, partition_on?)>
<!ELEMENT rule_ec ((column+|(lhs_ec, rhs_ec)), when?, join_on?)>
<!ELEMENT rule_cc (check, join_on?)>
<!ELEMENT if (condition|conditions)>
<!ELEMENT then (condition|conditions)>
<!ELEMENT when (condition|conditions)>
<!ELEMENT check (condition|conditions)>
<!ELEMENT join_on (column+)>
<!ELEMENT lhs (column_name+)>
<!ELEMENT rhs (column_name+)>
<!ELEMENT lhs_dd (distance_condition+)>
<!ELEMENT rhs_dd (distance_condition+)>
<!ELEMENT lhs_ec (column)>
<!ELEMENT rhs_ec (column|(table_name?, disj_attr+))>
<!ELEMENT partition_on (column_name+)>
```

```
<!ELEMENT column (table_name?, column_name)>
<!ELEMENT disj_attr (column_name+)>
<!ELEMENT distance_condition (column_name,
  comparison_operator, constant)>
<!ELEMENT conditions ((condition|conditions)+, boolean_operator)>
<!ELEMENT condition (lside, comparison_operator, rside)>
<!ELEMENT lside (column|arithmetic_computation)>
<!ELEMENT rside (constant|column|arithmetic_computation)>
<!ELEMENT arithmetic_computation ((constant|column|
  arithmetic_computation)+, arithmetic_operator)>
<!ELEMENT table_name (#PCDATA)>
<!ELEMENT column_name (#PCDATA)>
<!ELEMENT comparison_operator (#PCDATA)>
<!ELEMENT boolean_operator (#PCDATA)>
<!ELEMENT arithmetic_operator (#PCDATA)>
<!ELEMENT constant (#PCDATA)>
<!ATTLIST rule_definition name CDATA #IMPLIED
  type (conditional_rule|functional_dependency|
  order_dependency|distance_dependency|
  existence_constraint|check_constraint) #REQUIRED>
<!ATTLIST rule_od type (direct|inverse) #REQUIRED>
<!ATTLIST rule_ec type (ec_dep|ec_bidir|
  ec_disj|ec_attr) #REQUIRED> ]>
```

### XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="rules">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="rule_definition"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="rule_definition">
 <xs:complexType>
```

```
   <xs:sequence>
     <xs:element name="table_name" type="xs:string"
      minOccurs="1" maxOccurs="unbounded"/>
      <xs:choice>
       <xs:element ref="rule_cr"/>
       <xs:element ref="rule_fd"/>
       <xs:element ref="rule_od"/>
       <xs:element ref="rule_dd"/>
       <xs:element ref="rule_ec"/>
       <xs:element ref="rule_cc"/>
     </xs:choice>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string"/>
   <xs:attribute name="type" type="ruleType" use="required"/>
 </xs:complexType>
</xs:element>
<xs:simpleType name="ruleType">
 <xs:restriction base="xs:string">
   <xs:enumeration value="conditional_rule"/>
   <xs:enumeration value="functional_dependency"/>
   <xs:enumeration value="order_dependency"/>
   <xs:enumeration value="distance_dependency"/>
   <xs:enumeration value="existence_constraint"/>
   <xs:enumeration value="check_constraint"/>
 </xs:restriction>
</xs:simpleType>
<xs:element name="rule_cr">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="if"/>
     <xs:element ref="then"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="rule_fd">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="lhs"/>
```

```
      <xs:element ref="rhs"/>
      <xs:element ref="when" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="rule_od">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="lhs"/>
      <xs:element ref="rhs"/>
      <xs:element ref="when" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="partition_on" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="type" type="ocType" use="required"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="ocType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="direct"/>
    <xs:enumeration value="inverse"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="rule_dd">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="lhs_dd"/>
      <xs:element ref="rhs_dd"/>
      <xs:element ref="when" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="partition_on" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="rule_ec">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element ref="column" minOccurs="1" maxOccurs="2"/>
        <xs:sequence>
```

```
          <xs:element ref="lhs_ec"/>
          <xs:element ref="rhs_ec"/>
        </xs:sequence>
      </xs:choice>
      <xs:element ref="when" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="join_on" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="type" type="ecType" use="required"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="ecType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ec_dep"/>
    <xs:enumeration value="ec_bidir"/>
    <xs:enumeration value="ec_disj"/>
    <xs:enumeration value="ec_attr"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="rule_cc">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="check"/>
      <xs:element ref="join_on" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="if">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element ref="condition"/>
        <xs:element ref="conditions"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="then">
  <xs:complexType>
```

```
   <xs:sequence>
     <xs:choice>
       <xs:element ref="condition"/>
       <xs:element ref="conditions"/>
     </xs:choice>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="when">
 <xs:complexType>
   <xs:sequence>
     <xs:choice>
       <xs:element ref="condition"/>
       <xs:element ref="conditions"/>
     </xs:choice>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="check">
 <xs:complexType>
   <xs:sequence>
     <xs:choice>
       <xs:element ref="condition"/>
       <xs:element ref="conditions"/>
     </xs:choice>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="join_on">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="column"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="lhs">
 <xs:complexType>
```

```
   <xs:sequence>
     <xs:element name="column_name" type="xs:string"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="rhs">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="column_name" type="xs:string"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="lhs_dd">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="distance_condition"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="rhs_dd">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="distance_condition"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="lhs_ec">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="column"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="rhs_ec">
```

```
 <xs:complexType>
   <xs:sequence>
     <xs:choice>
      <xs:element ref="column"/>
      <xs:sequence>
        <xs:element name="table_name" type="xs:string"
         minOccurs="0" maxOccurs="1"/>
        <xs:element ref="disj_attr"
         minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:choice>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="partition_on">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="column_name" type="xs:string"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="column">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="table_name" type="xs:string"
      minOccurs="0" maxOccurs="1"/>
     <xs:element name="column_name" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="disj_attr">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="column_name" type="xs:string"
      minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
```

```
</xs:element>
<xs:element name="distance_condition">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="column_name" type="xs:string"/>
     <xs:element name="comparison_operator" type="xs:string"/>
     <xs:element name="constant"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="conditions">
 <xs:complexType>
   <xs:sequence>
     <xs:choice minOccurs="2" maxOccurs="2">
       <xs:element ref="condition"/>
       <xs:element ref="conditions"/>
     </xs:choice>
     <xs:element name="boolean_operator" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="condition">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="lside"/>
     <xs:element name="comparison_operator" type="xs:string"/>
     <xs:element ref="rside"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="lside">
 <xs:complexType>
   <xs:sequence>
     <xs:choice>
       <xs:element ref="column"/>
       <xs:element ref="arithmetic_computation"/>
     </xs:choice>
   </xs:sequence>
```

```
 </xs:complexType>
</xs:element>
<xs:element name="rside">
 <xs:complexType>
   <xs:sequence>
     <xs:choice>
       <xs:element name="constant" type="xs:string"/>
       <xs:element ref="column"/>
       <xs:element ref="arithmetic_computation"/>
     </xs:choice>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="arithmetic_computation">
 <xs:complexType>
   <xs:sequence>
     <xs:choice minOccurs="2" maxOccurs="2">
       <xs:element name="constant" type="xs:string"/>
       <xs:element ref="column"/>
       <xs:element ref="arithmetic_computation"/>
     </xs:choice>
     <xs:element name="arithmetic_operator" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

# Appendix B

## Schematron schema for OPM graphs

We used Schematron to validate the XML structure of the OPM graphs. In particular, we produced two versions of the Schematron schema considering the two definitions of legal OPM graph provided in literature (see the introduction to OPM in chapter 6). In the following, we show a unique Schematron schema containing all the defined assertions with, when necessary, comments specifying the corresponding legal definition.

In addition to Schematron assertions, the schema contains two XSLT functions used (as explained in chapter 7) to verity the presence of cycles in the graph.

```
<?xml version="1.0" encoding="utf-8"?>
<iso:schema
  xmlns="http://purl.oclc.org/dsdl/schematron"
  xmlns:iso="http://purl.oclc.org/dsdl/schematron"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fct="localFunctions"
  queryBinding="xslt2">
<iso:ns prefix="fct" uri="localFunctions"/>
<iso:title>Schematron schema for OPM graphs validation</iso:title>

<iso:pattern name="Nodes">
  <iso:rule context="artifact">
    <iso:assert test="@id=//dependencies/wasGeneratedBy/effect/@ref
     or @id=//dependencies/used/cause/@ref
     or @id=//dependencies/wasDerivedFrom/effect/@ref
     or @id=//dependencies/wasDerivedFrom/cause/@ref">
      An artifact is not connected to any process
```

```
      or any other artifact
    </iso:assert>
  </iso:rule>

  <iso:rule context="process">
    <iso:assert test="@id=//dependencies/used/effect/@ref
     or @id=//dependencies/wasGeneratedBy/cause/@ref
     or @id=//dependencies/wasTriggeredBy/effect/@ref
     or @id=//dependencies/wasTriggeredBy/cause/@ref">
      A process is not connected to any artifact
      or any other process
    </iso:assert>
  </iso:rule>

  <iso:rule context="dependencies">
    <iso:assert test="count(distinct-values(//agent/@id)) =
     count(distinct-values(wasControlledBy/cause/@ref))">
      An agent is not connected to any process
    </iso:assert>
  </iso:rule>
</iso:pattern>

<iso:pattern name="Edges">
  <iso:rule context="dependencies">
    <!-- Assertion for the OPM version 1.1 -->
    <iso:assert test="count(wasGeneratedBy) =
     count(wasGeneratedBy[not(effect/@ref =
     preceding-sibling::wasGeneratedBy/effect/@ref)])">
      At most one wasGeneratedBy edge per artifact is allowed
    </iso:assert>

    <!-- Assertion for the "second" definiton of OPM legal graph -->
    <iso:assert test="count(wasGeneratedBy[role]) =
     count(wasGeneratedBy[role and not(effect/@ref =
     preceding-sibling::wasGeneratedBy/effect/@ref)])">
      At most one precise wasGeneratedBy edge
      per artifact is allowed
    </iso:assert>
```

```
  </iso:rule>

  <iso:rule context="wasTriggeredBy">
    <iso:assert test="not(effect/@ref=cause/@ref)">
      The two processes connected by a wasTriggeredBy edge
      have to be distinct
    </iso:assert>
  </iso:rule>

  <iso:rule context="wasDerivedFrom">
    <iso:assert test="not(effect/@ref=cause/@ref)">
      The two artifacts connected by a wasDerivedFrom edge
      have to be distinct
    </iso:assert>

    <!-- Assertion for the "second" definiton of OPM legal graph -->
    <let name="a1ID" value="effect/@ref"/>
    <let name="a2ID" value="cause/@ref"/>
    <iso:assert test="role and
     ../wasGeneratedBy[role and effect/@ref=$a1ID]/cause/@ref =
     ../used[role and cause/@ref=$a2ID]/effect/@ref and
     ../used[cause/@ref=$a2ID]/role/@id=role/@id">
      For every precise wasDerivedFrom edge connecting
      artifact A1 and artifact A2, a wasGeneratedBy edge
      connecting A1 with a process and a used edge
      connecting the same process to A2 are necessary
    </iso:assert>
  </iso:rule>
</iso:pattern>

<iso:pattern name="Time">
  <iso:rule context="wasControlledBy">
    <let name="processID" value="effect/@ref"/>
    <iso:assert test="(
     (../wasGeneratedBy[cause/@ref=$processID]/time/@exactlyAt >
     startTime/@exactlyAt and endTime/@exactlyAt >
     ../wasGeneratedBy[cause/@ref=$processID]/time/@exactlyAt)
     or not(../wasGeneratedBy/cause/@ref=$processID) )
```

```
     and (
     (../used[effect/@ref=$processID]/time/@exactlyAt >
     startTime/@exactlyAt and endTime/@exactlyAt >
     ../used[effect/@ref=$processID]/time/@exactlyAt)
     or not(../used/effect/@ref=$processID) )">
      Timestamps assigned to the edges are not consistent
    </iso:assert>

    <iso:assert test="(endTime/@exactlyAt) > (startTime/@exactlyAt)">
      Timestamps assigned to a wasControlledBy edge
      are not consistent
    </iso:assert>
  </iso:rule>

  <iso:rule context="wasGeneratedBy">
    <let name="artifactID" value="effect/@ref"/>
    <iso:assert test="../used[cause/@ref=$artifactID]/time/@exactlyAt
     > time/@exactlyAt or not(../used/cause/@ref=$artifactID)">
      Timestamps assigned to wasGeneratedBy and used
      edges are not consistent
    </iso:assert>
  </iso:rule>

  <iso:rule context="used">
    <let name="artifactID" value="cause/@ref"/>
    <iso:assert test="time/@exactlyAt >
     ../wasGeneratedBy[effect/@ref=$artifactID]/time/@exactlyAt
     or not(../wasGeneratedBy/effect/@ref=$artifactID)">
      Timestamps assigned to used and wasGeneratedBy
      edges are not consistent
    </iso:assert>
  </iso:rule>
</iso:pattern>

<iso:pattern name="Cycles">
  <iso:rule context="dependencies">
    <iso:assert test="fct:checkCycleTest(wasDerivedFrom)">
      For a single account, wasDerivedFrom cycles are not allowed
```

```
    </iso:assert>
  </iso:rule>
</iso:pattern>

<xsl:function name="fct:checkCycleTest" as="xs:boolean">
  <xsl:param name="wdfs" as="node()*"/>
<xsl:choose>
    <xsl:when test="count($wdfs)>0">
      <xsl:variable name="wdfe1">
        <xsl:value-of select="$wdfs[position()=1]/effect/@ref"/>
      </xsl:variable>
      <xsl:variable name="wdfc1">
        <xsl:value-of select="$wdfs[position()=1]/cause/@ref"/>
      </xsl:variable>
      <xsl:variable name="account">
        <xsl:value-of select="$wdfs[position()=1]/account/@ref"/>
      </xsl:variable>
    <xsl:choose>
        <xsl:when test="fct:checkCycle($wdfs,$wdfe1,
                                       $wdfc1,$account,$wdfs)">
      <xsl:value-of select="
      fct:checkCycleTest($wdfs[position()>1])"/>
        </xsl:when>
      <xsl:otherwise>
      <xsl:value-of select="false()"/>
      </xsl:otherwise>
    </xsl:choose>
    </xsl:when>
  <xsl:otherwise>
  <xsl:value-of select="true()"/>
  </xsl:otherwise>
</xsl:choose>
</xsl:function>

<xsl:function name="fct:checkCycle" as="xs:boolean">
  <xsl:param name="wdfs" as="node()*"/>
  <xsl:param name="wdfe1" as="xs:string"/>
  <xsl:param name="wdfc1" as="xs:string"/>
```

```
  <xsl:param name="account" as="xs:string"/>
  <xsl:param name="wdfsbis" as="node()*"/>
<xsl:choose>
    <xsl:when test="count($wdfs)>0">
      <xsl:choose>
        <xsl:when test="$wdfs[position()=1]/effect/@ref=$wdfc1 and
                  $wdfs[position()=1]/account/@ref=$account">
         <xsl:choose>
            <xsl:when test="$wdfs[position()=1]/cause/@ref=$wdfe1">
             <xsl:value-of select="false()"/>
            </xsl:when>
            <xsl:otherwise>
       <xsl:variable name="wdfid">
          <xsl:value-of select="$wdfs[position()=1]/@id"/>
       </xsl:variable>
            <xsl:value-of select="
            fct:checkCycle($wdfsbis[not(@id=$wdfid)],$wdfe1,
            $wdfs[position()=1]/cause/@ref,$account,$wdfsbis)"/>
       </xsl:otherwise>
          </xsl:choose>
        </xsl:when>
        <xsl:otherwise>
       <xsl:value-of select="
       fct:checkCycle($wdfs[position()>1],
           $wdfe1,$wdfc1,$account,$wdfsbis)"/>
        </xsl:otherwise>
      </xsl:choose>
     </xsl:when>
     <xsl:otherwise>
       <xsl:value-of select="true()"/>
     </xsl:otherwise>
  </xsl:choose>
</xsl:function>
</iso:schema>
```

# Appendix C

## Information Product Map

The Information Product Map (IP-MAP) graphical model is centered on the concept of information product, which is a collection of data element instances that meet the specified requirements of a data consumer. An IP-MAP is aimed at being a systematic representation of the details associated with the creation of such a product.

An information product (IP) is produced by means of processing activities and data quality checks on raw data and semi-processed information (or component data). Raw data (RD) are defined as data that come from outside the boundaries of the IP-MAP. Component data (CD) are defined as a set of temporary semi-processed information that might be generated within the IP-MAP and used in creating the final IP.

The following eight construct blocks can be used to build an IP-MAP:

- Source (input data) block: it represents the source of each raw data that must be available in order to produce the IP.
- Customer (output) block: it is used to represent the consumers of the IP.
- Data Quality (evaluation) block: associated with this block there is a list of the data quality checks that are performed on the specified component data items; this block has two possible output streams, correct and incorrect, depending on the result of the evaluation of the checks.
- Processing block:
    1. Primary purpose - it is used to represent any manipulation or calculation involving some or all of the raw input or component data items required to ultimately produce the IP.
    2. Secondary purpose - it is used as a data correction block: when errors are identified in a set of data items that enter this block, some corrective

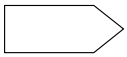
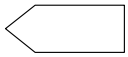
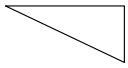
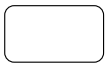
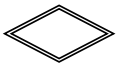
| Construct name | Construct symbol |
|---|---|
| Source (input data) Block |  |
| Customer (output) Block |  |
| Data Quality (evaluation) Block |  |
| Processing Block |  |
| Decision Block |  |
| Data Storage Block |  |
| Organizational Boundary Block |  |
| Information System Boundary Block |  |

**Table 1:** IP-MAP construct symbols

action is required; this block represents a process that is not part of the standard processing sequence, but is utilized under special circumstances.

– Decision block: when it is necessary to direct the data items to a different set of blocks for further processing, a decision block captures the different conditions to be evaluated and the corresponding procedures for handling the incoming data items based on the evaluation.

– Data Storage block: it is used to represent data items (raw and component data), stored in storage files or databases, that wait for further processing or are captured as part of the information inventory in the organization.

– Organizational Boundary block: it is used when a data unit (raw or component data) moves across departments or across organizations.

– Information System Boundary block: it is used to reflect the changes to a data unit (raw or component data) as it moves from one information system (paper or computerized) to another type of information system (paper or comput-

erized). The information system boundary block is used to specify the two information systems involved.

Each construct block is identified by a unique name and is further described by a set of attributes (i.e., metadata). Table 1 lists the graphical symbols assigned to each construct block.

The different types of data involved in every step of the process can be differentiated in the diagram by means of labels (i.e., IP, RD, and CD respectively for information products, row data, and component data) on the arrows connecting each pairs of construct blocks.

# Bibliography


M. K. Anand, S. Bowers, T. McPhillips, and B. Ludäscher. Efficient provenance storage over nested data collections. In M. Kersten et al., editors, *Proceedings of the 12th Int'l Conference on Extending Database Technology (EDBT2009)*, volume 360 of *AICPS*, pages 958–969. ACM, 2009.

M.M. Aqel, N.F. Shilbayeh, and M.S. Hakawati. CFD-Mine: An efficient algorithm for discovering functional and conditional functional dependencies. *Trends in Applied Sciences Research*, 7(4):285–302, 2012.

P. Atzeni and N.M. Morfuni. Functional dependencies and constraints on null values in database relations. *Information and Control*, 70(1):1–31, 1986.

M. Bajec, M. Krisper, and R. Rupnik. Using business rules technologies to bridge the gap between business and business applications. In G. Rechnu, editor, *Proceedings of the 16th IFIP World Computer Congress*, pages 77–85, 2000.

J. Barateiro and H. Galhardas. A survey of data quality tools. *Datenbank-Spektrum*, 14/2005:15–21, 2005.

C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer-Verlag, 2006.

C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Computing Surveys*, 41(3), 16, 2009.

M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *Journal of Compututer and System Sciences*, 59(1):94–115, 1999.

C. Beeri and M. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

M. Bergdahl, M. Ehling, E. Elvers, E. Földesi, T. Körner, A. Kron, P. Lohauß, K. Mag, V. Morais, A. Nimmergut, A.V. Sæbø, U. Timm, and M. João Zilhão.

*Handbook on Data Quality Assessment Methods and Tools (DatQAM)*. Eurostat, 2007.

L. Bertossi and J. Chomicki. Query answering in inconsistent databases. In J. Chomicki, G. Saake, and R. van der Meyden, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2003.

A. Bilke, J. Bleiholder, C. Böhm, K. Draba, F. Naumann, and M. Weis. Automatic data fusion with HumMer. In K. Böhm et al., editors, *Proceedings of the 31st Int'l Conference on Very Large Data Bases (VLDB2005)*, pages 1251–1254. ACM, 2005.

O. Biton, S. Cohen-Boulakia, S.B. Davidson, and C. Hara. Querying and managing provenance through user views in scientific workflows. In *Proceedings of the 24th IEEE Int'l Conference on Data Engineering (ICDE2008)*, pages 1072–1081, 2008.

J. Bleiholder and F. Naumann. Declarative data fusion – syntax, semantics, and implementation. In *Proceedings of the 9th East European conference on Advances in Databases and Information Systems (ADBIS2005)*, pages 58–73, 2005.

J. Bleiholder, K. Draba, and F. Naumann. FuSem – exploring different semantics of data fusion. In C. Koch et al., editors, *Proceedings of the 33rd Int'l Conference on Very Large Data Bases (VLDB2007)*, pages 1350–1353. ACM, 2007.

P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the 23th IEEE Int'l Conference on Data Engineering (ICDE2007)*, pages 746–755, 2007.

M. Bovee, R. P. Srivastava, and B. Mak. A conceptual framework and belief-function approach to assessing overall information quality. *Int'l Journal of Intelligent Systems*, 18:51–74, 2003.

L. Bravo, W. Fan, F. Geerts, and S. Ma. Increasing the expressivity of conditional functional dependencies without extra complexity. In *Proceedings of the 24th IEEE Int'l Conference on Data Engineering (ICDE2008)*, pages 516–525, 2008.

P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–12. ACM, 2008.

W. Chen, W. Fan, and S. Ma. Analyses and validation of conditional dependencies with built-in predicates. In S.S. Bhowmick, J. Kung, and R. Wagner, editors, *Proceedings of the 20th Int'l Conference on Database and Expert Systems Applications (DEXA09)*, volume 5690 of *LNCS*, pages 576–591. Springer-Verlag, 2009.

Y. Chen. Quality of fisheries data and uncertainty in Stock Assessment. *Scientia Marina*, 67(Suppl. 1):75–87, 2003.

J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2007.

F. Chiang and R.J. Miller. Discovering data quality rules. *Proceedings of the VLDB Endowment*, 1(1):1166–1177, 2008.

J. Chomicki. Consistent query answering: Opportunities and limitations. In S. Bressan, J. Küng, and R. Wagner, editors, *Proceedings of the 17th Int'l Conference on Database and Expert Systems Applications (DEXA06)*. Springer-Verlag, 2006.

P. Christen. Febrl: An open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining (KDD08)*, pages 1065–1068, 2008.

P. Christen. Development and user experiences of an open source data cleaning, deduplication and record linkage system. *ACM SIGKDD Explorations Newsletter*, 11(1):39–48, 2009.

G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In C. Koch et al., editors, *Proceedings of the 33rd Int'l Conference on Very Large Data Bases (VLDB2007)*, pages 315–326. ACM, 2007.

A. Cooper. *Guide to Fisheries Stock Assessment: From data to recommendations*. University of New Hampshire/NH Sea Grant, 2006.

T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley, 2003.

S.B. Davidson, S. Cohen Boulakia, A. Eyal, B. Ludäscher, T.M. McPhillips, S. Bowers, M.K. Anand, and J. Freire. Provenance in scientific workflow systems. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 30(4): 44–50, 2007.

L. Ding, J. Futrelle, D. Garijo Verdejo, P. Groth, M. Jewell, S. Miles, P. Missier, J. Pan, and J. Zhao. Open Provenance Model (OPM) OWL specification, 2010. Available as `http://openprovenance.org/model/opmo`.

M.G. Elfeky, A.K. Elmagarmid, and V.S. Verykios. TAILOR: A record linkage tool box. In *Proceedings of the 18th IEEE Int'l Conference on Data Engineering (ICDE2002)*, pages 17–28, 2002.

A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

R. Elmasri and S.B. Navathe. *Foundamentals of Database Systems (3rd edition)*. Addison-Wesley, 2000.

European Commission. Commission Regulation (EC) No 1639/2001 of 25 July 2001. *Official Journal of the European Union*, 17 August 2001.

European Commission. Commission Decision 2005/629/EC of 26 August 2005. *Official Journal of the European Union*, 31 August 2005.

European Commission. Commission Regulation (EC) No 665/2008 of 14 July 2008. *Official Journal of the European Union*, 15 July 2008.

European Commission. *Communication from the Commission concerning a consultation on Fishing Opportunities. COM/2011/298 final*, 2011.

European Commission. Regulation (EC) No 223/2009 of the European Parliament and of the Council of 11 March 2009. *Official Journal of the European Union*, 31 March 2009.

W. Fan, F. Geerts, and X. Jia. Semandaq: A data quality system based on conditional functional dependencies. *Proceedings of the VLDB Endowment*, 1(2): 1460–1463, 2008a.

W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems (TODS)*, 33(2):94–115, 2008b.

W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(5): 683–697, 2011.

I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th Int'l Conference on Scientific and Statistical Database Management (SSDBM)*, pages 1–10, 2002.

A. Frank and A. Asuncion. *UCI Machine Learning Repository*, 2010. URL `http://archive.ics.uci.edu/ml`.

C. Fürber and M. Hepp. Using SPARQL and SPIN for data quality management on the Semantic Web. In W. Abramowicz and R. Tolksdorf, editors, *Proceedings of the 13th Int'l Conference on Business Information Systems (BIS2010)*, volume 47 of *Lecture Notes in Business Information Processing*, pages 35–46, 2005.

H. Galhardas, D. Florescu, D. Shasha, and E. Simon. AJAX: An extensible data cleaning tool. In W. Chen, J.F. Naughton, and P.A. Bernstein, editors, *Proceedings of the ACM SIGMOD Int'l Conference on Management of Data*, pages 1065–1068, 2000.

Y. Gil, J. Cheney, P. Groth, O. Hartig, S. Miles, L. Moreau, and P. Pinheiro da Silva. Final report of the W3C Provenance Incubator Group, 2010. Available as `http://www.w3.org/2005/Incubator/prov/wiki/Final_Report_Draft`.

S. Ginsburg and R. Hull. Order dependency in the relational model. *Theoretical Computer Science*, 26:149–195, 1983.

B. Glavic and K. Dittrich. Data provenance: A categorization of existing approaches. *BTW*, 1(4):227–241, 2007.

L. Golab, H.J. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *Proceedings of the VLDB Endowment*, 1(1):376–390, 2008.

L. Golab, H.J. Karloff, F. Korn, and D. Srivastava. Data Auditor: Exploring data quality and semantics using pattern tableaux. *Proceedings of the VLDB Endowment*, 3(2):1641–1644, 2010.

Q. Görz and M. Kaiser. An indicator function for insufficient data quality: A contribution for data accuracy. In H. Rahman, A. Mesquita, I. Ramos, and B. Pernici, editors, *Proceedings of the 7th Mediterranean Conference on Information Systems*, volume 129 of *Lecture Notes in Business Information Processing*, pages 169–184, 2012.

P. Groth and L. Moreau. The Open Provenance Model XML Schema, 2010. URL `http://openprovenance.org/model/opmx-20101012.xsd`.

L. Gu, R. Baxter, D. Vickers, and C. P. Rainsford. Record linkage: Current practice and future directions. CMIS technical report 03/83, CSIRO Mathematical and Information Sciences, Camberra, Australia, 2003.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

E.R. Harold and W.S. Means. *XML in a Nutshell*. O'Reilly Media, 2004.

J.M. Hellerstein. Quantitative data cleaning for large databases, 2008. Report for the United Nations Economic Commission for Europe (UNECE).

J. Hipp, U. Güntzer, and U. Grimmer. Data quality mining – Making a virtue of necessity. In *Proceedings of the 6th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD01)*, pages 52–57. ACM, 2001.

J.A. Hoffer, M.B. Prescott, and F.R. McFadden. *Modern Database Management (8th edition)*. Prentice Hall, New York, 2007.

Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal*, 42(2):100–111, 1999.

J.M. Juran. *Managerial Breakthrough*. McGraw-Hill, New York, 1964.

W. Kim, B.-J. Choi, E.-K. Hong, S.-K. Kim, and D. Lee. A taxonomy of dirty data. *Data Mining and Knowledge Discovery*, 7:81–99, 2003.

N. Kwasnikowska, L. Moreau, and J. Van den Bussche. A formal account of the Open Provenance Model, 2010. Available as `http://eprints.soton.ac.uk/271819`.

M. Lee, T. Ling, and W. Low. IntelliClean: A knowledge-based intelligent data cleaner. In *Proceedings of the 6th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining (KDD00)*, pages 290–294. ACM, 2000.

Y.W. Lee, D.M. Strong, B.K. Kahn, and R.Y. Wang. AIMQ: A methodology for information quality assessment. *Information and Management*, 40(2):133–146, 2002.

Y.W. Lee, L.L. Pipino, J.D. Funk, and R.Y. Wang. *Journey to Data Quality*. The MIT Press, 2006.

C. Li. The Flamingo software package on approximate string queries. In J. Xu, G. Yu, S. Zhou, and R. Unland, editors, *Database Systems for Adanced Applications, 16th Int'l Conference DASFAA, Int'l Workshops*, volume 6637 of *LNCS*. Springer-Verlag, 2011.

J. Li, H. Liu, J. Toivonen, and J. Yong. Effective pruning for the discovery of conditional functional dependencies. *The Computer Journal*, 2012. Accepted for publication.

C. Lim, S. Lu, A. Chebotko, and F. Fotouhi. Storing, reasoning, and querying OPM-compliant scientific workflow provenance using relational databases. *Future Generation Computer Systems*, 27:781–789, 2011.

J. Liu, J. Li, C. Liu, and Y. Chen. Discover dependencies from data – A review. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(2):251–264, 2012.

S. Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and Armstrong relations. In C. Zaniolo, P.C. Lockemann, M.H. Scholl, and T. Grust, editors, *Proceedings of the 7th Int'l Conference on Extending Database Technology (EDBT2000)*, volume 1777 of *LNCS*, pages 350–364. Springer-Verlag, 2000.

M.J. Maher. Constrained dependencies. *Theoretical Computer Science*, 173:113–149, 1997.

H. Mannila and K. Raiha. Design by example: An application of Armstrong relations. Technical report, Cornell University Ithaca, 1985.

I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad, editors, *Proceedings of the 12th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining (KDD06)*, pages 935–940. ACM, 2006.

S. Miles, P. Groth, S. Munroe, and L. Moreau. PrIMe: A methodology for developing provenance-aware applications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(3), 8, 2011.

P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C.A. Goble. Data lineage model for Taverna workflows with lightweight annotation requirements. In J. Freire, D. Koop, and L. Moreau, editors, *Provenance and Annotation of Data and Processes, 2nd Int'l Provenance and Annotation Workshop (IPAW2008)*, volume 5272 of *LNCS*, pages 17–30. Springer-Verlag, 2008.

L. Moreau, J. Freire, J. Futrelle, R.E. McGrath, J. Myers, and P. Paulson. The Open Provenance Model (v1.00). Technical report, University of Southampton, 2007. Available as `http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf`.

L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmahan, E. Stephan, and J. Van den

Bussche. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, 27:743–756, 2011.

H. Müller and J.-C. Freytag. Problems, methods and challenges in comprehensive data cleansing. Technical report HUB-IB-164, Humboldt-Universität zu Berlin, Institut für Informatik, 2003.

F. Naumann. *Quality-Driven Query Answering for Integrated Information Systems*, volume 2261 of *LNCS*. Springer-Verlag, 2002.

W. Ng. Order functional dependencies in relational databases. *Information Systems*, 24(7):535–554, 1999.

OECD. *Handbook on Construction Composite Indicators – Methodology and User Guide*. Organization for Economic Co-operation and Development, 2008.

P. Oliveira, F. Rodrigues, and P. Henriques. A formal definition of data quality problems. In F. Naumann, M. Gertz, and S.E. Madnick, editors, *Proceedings of the 10th Int'l Conference on Information Quality (ICIQ2005)*. MIT, 2005a.

P. Oliveira, F. Rodrigues, P. Henriques, and H. Galhardas. A taxonomy of data quality problems. In *Proceedings of the 2nd Int'l Workshop on Data and Information Quality (in conjunction with CAiSE05)*, pages 452–457, 2005b.

P. Oliveira, F. Rodrigues, and P. Henriques. SmartClean: An incremental data cleaning tool. In B. Choi, editor, *Proceedings of the 9th Int'l Conference on Quality Software*, pages 452–457, 2009.

K. Orr. Data quality and systems theory. *Communications of the ACM*, 2:66–71, 1998.

L.L. Pipino, Y.W. Lee, and R.Y. Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002.

O. Pivert and H. Prade. Handling dirty databases: From user warning to data cleaning – Towards an interactive approach. In A. Deshpande and A. Hunter, editors, *Proceedings of the 4th Int'l Conference on Scalable Uncertainty Management (SUM2010)*, volume 6379 of *LNAI*, pages 292–305. Springer-Verlag, 2010.

E. Rahm and H. Do. Data cleaning: Problems and current approaches. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 23(4): 3–13, 2000.

V. Raman and J. Hellerstein. Potter's Wheel: An interactive data cleaning system. In P.M.G. Apers et al., editors, *Proceedings of the 27th Int'l Conference on Very Large Data Bases (VLDB2001)*, pages 381–390. Morgan Kaufmann Publishers Inc., 2001.

T.C. Redman. *Data Quality – The Field Guide*. Digital Press, Boston, MA, 2001.

K.-U. Sattler and E. Schallehn. A data preparation framework based on a multidatabase language. In *Proceedings of the 5th Int'l Database Engineering & Applications Symposium (IDEAS01)*, pages 1065–1068, 2001.

M. Scannapieco and T. Catarci. Data quality under the computer science perspective. *Archivi & Computer*, 2, 2002.

G. Shankaranarayan and R. Y. Wang. IPMAP: Current state and perspectives. In *Proceedings of the 12th Int'l Conference on Information Quality (ICIQ2007)*, pages 1–18. MIT, 2007.

G. Shankaranarayan, R. Y. Wang, and M. Ziad. Modeling the manufacture of an information product with IP-MAP. In *Proceedings of the 5th Int'l Conference on Information Quality (ICIQ2000)*, pages 1–16. MIT, 2000.

G. Shanks and B. Corbitt. Understanding data quality: Social and cultural aspects. In *Proceedings of the 10th Australasian Conference on Information Systems*, pages 785–797, 1999.

Y. L. Simmhan, B. Plale, and D. Gannon. Karma2: Provenance management for data-driven workflows. *Int'l Journal of Web Services Research*, 5(2):1–22, 2008.

S. Song and L. Chen. Differential dependencies: Reasoning and discovery. *ACM Transactions on Database Systems (TODS)*, 26(3), 16, 2011.

J. Szlichta, P. Godfrey, and J. Gryz. Fundamentals of order dependencies. *Proceedings of the VLDB Endowment*, 5(11):1120–1231, 2012.

W.-C. Tan. Provenance in databases: Past, current, and future. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 30(4):3–12, 2007.

K. Taveter and G. Wagner. Agent-oriented enterprise modeling based on business rules. In H.S. Kunii, S. Jajodia, and A. Sölvberg, editors, *Proceedings of 20th Int'l Conference on Conceptual Modeling (ER2001)*, volume 2224 of *LNCS*, pages 527–540. Springer-Verlag, 2001.

E. van der Vlist. *Schematron.* O'Reilly Media, 2007.

P. Vassiliadis. A survey of extract-transform-load technology. *Int'l Journal of Data Warehousing & Mining*, 5(3):1–27, 2009.

P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis. Arktos: Towards the modeling, design, control and execution of ETL processes. *Information Systems*, 26(8):537–561, 2001.

R.Y. Wang. A product perspective on total data quality management. *Communications of the ACM*, 41(2):58–65, 1998.

R.Y. Wang, V.C. Storey, and C.P. Firth. A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 7(4): 623–640, 1995.

C.M. Wyss, C. Giannella, and E.L. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In Y. Kambayashi, W. Winiwarter, and M. Arikawa, editors, *Proceedings of the 3rd Int'l Conference on Data Warehousing and Knowledge Discovery (DaWak2001)*, volume 2114 of *LNCS*, pages 101–110. Springer-Verlag, 2001.

H. Yao and H.J. Hamilton. Mining functional dependencies from data. *Journal Data Mining and Knowledge Discovery*, 16(2):197–219, 2008.

H. Yao, H.J. Hamilton, and C.J. Butz. FD_Mine: Discovering functional dependencies in a database using equivalences. In V. Kumar et al., editors, *Proceedings of the 2nd IEEE Int'l Conference on Data Mining (ICDM2002)*, pages 729–732, 2002.

J. Zhao, Y. Simmhan, K. Gomadam, and V. Prasanna. Querying provenance information in distributed environments. *Int'l Journal of Computers and Their Applications (IJCA)*, 18(3):196–215, 2011.