Early Integration Testing for Entity Reconciliation in the Context of Heterogeneous Data Sources

Raquel Blanco, José G. Enríquez, Francisco J. Domínguez-Mayo, M. J. Escalona, and Javier Tuya, *Member, IEEE*

***Abstract*—Entity reconciliation (ER) aims to combine data from different sources for a unified vision. The management of large volumes of data has given rise to significant challenges to the ER problem due to facts such as data becoming more unstructured, unclean, and incomplete or the existence of many datasets that store information about the same topic. Testing the applications that implement the ER problem is crucial to ensure both the correctness of the reconciliation process and the quality of the reconciled data. This paper presents an approach based on model-driven engineering that allows the creation of test models for the early integration testing of ER applications, contributing in three main aspects: the description of the elements of the proposed framework, the definition of the testing model, and the validation of the proposal through two real-world case studies. This validation verifies that the early integration testing of the ER application is capable of detecting a series of deficiencies, which *a priori* are not known and that will help to improve the final result that the ER application offers.**

***Index Terms*—Early testing, entity reconciliation, heterogeneous data sources, model-driven engineering, software testing, specification-based testing.**

Nomenclature

ER	Entity reconciliation.
ICT	Information and Communications Technology.
MDE	Model-driven engineering.
ETL	Extract, transform and load.
ITR	Integration testing rules.
SBVR	Sematics of Business Vocabulary and Business Rules.
EBNF	Extended Backnus-Naur Form.

This work was supported in part by the Spanish Ministry of Science and Technology, under the projects PERTEST (TIN2013-46928-C3-1-R), MeGUS (TIN2013-46928-C3-3-R), TestEAMos (TIN2016-76956-C3-1-R), POLOLAS (TIN2016-76956-C3-2-R), and SoftPLM Network (TIN2015-71938-REDT); in part by the Principality of Asturias (Spain), under the project GRUPIN14-007; in part by European Regional Development Fund; and in part by Fujitsu Laboratories of Europe. Associate Editor: P. Laplante. *(Corresponding author: Raquel Blanco.)*

R. Blanco and J. Tuya are with the Department of Computer Science, University of Oviedo, Oviedo 33003, Spain (e-mail: rblanco@uniovi.es; tuya@uniovi.es).

J. G. Enríquez, F. J. Domínguez-Mayo, and M. J. Escalona are with the Department of Computer and Language Systems, University of Seville, Seville 41004, Spain (e-mail: jose.gonzalez@iwt2.org; fjdominguez@us.es; mjescalona@us.es).

I. Introduction

CURRENTLY, information management is critical in many aspects of our lives. However, the incorporation of ICT into everyday life causes people to experience an overload of information, also known by the term "infoxication." This term refers to the difficulty that someone has in understanding a problem and making decisions about it because of an excess of information [1].

In the first era of ICT, the main problem that researchers had was how to find information and how to store and manage it efficiently. Currently, due to the existence of Big Data and cloud computing, the biggest problem is how to extract knowledge from the information depending on the needs of each user [2]. Considering the large number of data sources that store information related to the same topic, the need for heterogeneity and cross-domain reconciliation become important features. In this context, the problem of ER plays an important role in data management, being one of the major research problems in data quality management [3].

ER (also called entity resolution) is a fundamental problem in data integration. It refers to combining data from different sources for a unified vision or, in other words, identifying entities from the digital world that refer to the same real-world entity. It is an uncertain process because the decision to allocate a set of records with the same entity cannot be taken with certainty unless these records are identical in all their attributes or they have a common key [4], [5]. This problem can be applied to many different scenarios such as terrorist screening, insurance fraud detection or e-health environments, among others.

While this problem is not new, the management of large volumes of data presents new challenges and the necessity of carrying out high-quality reconciliation of entities continues to grow in the era of Big Data [2], [6]. Getoor and Machanavajjhala [7] expose some of the main challenges of the ER in the Big Data environment such as:

1) data heterogeneity, where it is becoming more common that data are unstructured, unclean, or incomplete and also there are diverse data types;
2) data being more linked, where it is necessary to infer relationships in addition to equality;
3) making multirelational data, dealing with structure of entities;
4) building multidomain systems, trying to customize methods that span across domains.

Due to the important challenges of the ER problem, it is crucial to test the operations designed to carry out the reconciliations and the applications that implement them, in order to ensure both the correctness of the reconciliation and the high quality of the reconciled data.

In this paper, we propose an approach based on the MDE paradigm for testing applications that implement ER problems. This approach relies on the ER problem specification and the conceptual data models of the sources and the solution to be achieved in order to define test models composed of a set of business rules, which specify the system requirements. From these business rules, the situations of interest to be tested (*test coverage items*) can be automatically derived to guide the generation of the test cases.

MDE [8] emerged to address the complexity of software systems in order to express the concepts of the problem domain in an effective way. In this way, the basic principle of MDE is "Everything is a model" [9]. The main idea of the MDE is to use a set of models to decrease the level of abstraction. Thus, in the early stages of development, models are more abstract than in the final stages where the models are much closer to implementation. One of the advantages of MDE is its support for automation, as the models can be automatically transformed from the early stages of development to the final stages. Therefore, MDE allows automating the tasks involved in software development, such as the testing tasks.

In an earlier work [10], a first approach based on MDE that allows the creation of test models for the integration testing of ER applications was presented. In this new work, the test model for integration testing, called the ITR model, is developed in depth. In addition, we describe the application of the ITR model to two real-world problems. The main contributions of this work are as follows.

1) The description of the elements that constitute the framework for testing the ER applications.
2) The definition of the ITR model for integration testing, which represents the testing objectives as a set of business rules, called *integration rules*.
3) The application of the proposal to two real-world problems.

The remainder of this paper is organized as follows: Section II provides background and related work. Section III formulates the ER testing problem. Sections IV and V describe the framework for testing ER applications and the ITR model. Section VI presents a real-world case study. Finally, this paper ends with conclusions and future work.

II. Background and Related Work

ER is a well-known problem and it has been investigated since the birth of relational databases. With the advent of Big Data, it has received significant attention due to the new challenges that arise as mentioned above. The techniques for solving this kind of problem can be broadly classified into: deterministic rule-based methods [11]–[13], probabilistic-based methods [14]–[17], learning based techniques [18]–[21], and graph-based techniques [22]–[26].

The approach in which early testing has been integrated is the one presented in [2], where authors proposed an ER approach based upon MDE and virtual graphs. This approach has some relation with ETL [27] although the main goal of these kinds of tools is not the development of the ER process but the integration of information from different data sources into one or legacy systems integrations. It has been very difficult to find related work about testing in ER: however, taking into account that the proposal bears some resemblance to the ETL, some works are presented.

A variety of works can be found in the literature about testing ETL processes. Some of them are related to analyzing the impact of automated ETL testing on the data quality or to evaluating the quality of different approaches [28], [29]. Dakrory *et al.* [30] proposed a testing framework to automate testing data quality at the stage of the ETL process by automating the creation and execution of these tests. Tesfagiorgish and JunYi [31] proposed an approach of big data transformation testing based on the concept of data reverse engineering. The closest work that has been found is the one presented in [32]. The authors developed a test framework that generates a small and representative dataset from an original large dataset using input space partition testing. However, this paper proposes using early testing in the ER process, and as the approach developed is not an ETL system, the objectives of the two papers are different.

Early testing focuses on the first phases of the software development lifecycle [33]. One of the reasons for integrating early testing in the selected approach is the benefits that it produces in reducing costs in the verification and validation phase, and the reduction of its complexity [34]. Most of the works related to early testing study the automation of test case generation [35]–[38]. The present work differs from foregoing works in that it is not based on test case generation, but on the automation of test coverage items that will guide the generation of test cases.

III. Problem Statement

Consider, for example, the following scenario: the information stored into two databases DB1 and DB2, composed of the tables R and S, respectively, is going to be reconciled into a graph structure. Each row r^i of *R* and s^j of S is considered an entity. The information that will constitute the solution of the ER process, called the *reconciled solution*, is represented in nodes and edges, where each node is an entity and the edges are relationships between entities. The software engineer defines the conceptual data model of this reconciled solution (henceforth *reconciled solution model*), which contains the types of nodes (that is, the types of entities) T and U, as well as the type of edge (that is, the type of relationship) V.

Fig. 1 depicts the schemas of the data sources (DB1 and DB2) and the model of the reconciled solution. The attributes C1, D1, and D2 do not uniquely identify the entities t^k of T and u^m of U.

According to the reconciliation specification, an entity r^i of R is represented in the reconciled solution by some related entities t^k of T and u^m of U. An entity s^j of S is also represented by some related entities of T and U. Besides, an entity r^i and

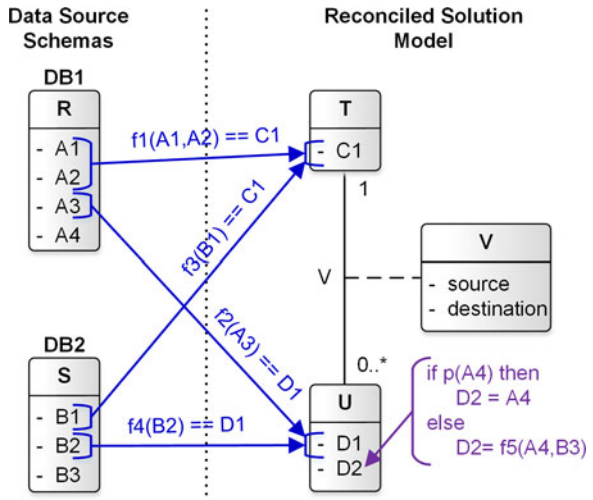

Fig. 1. Introductory example.

an entity s^j could be represented by the same related entities of T and U. The projection from R and S to the reconciled solution is carried out via functions over their attributes (for example, f1(A1,A2) leads the projection from an entity r^i to entities t^k when the result of its evaluation is equal to C1). The reconciliation specification also indicates that the value of the attribute D2 is derived from the entities of R and S that correspond to the same related entities of T and U: D2 takes its value from A4 only if a predicate p(A4) is found to hold true; otherwise, it takes its value from the function f5(A4, B3).

The left side of Fig. 2 shows an example of the information stored in the data sources, which is going to be reconciled by an application that implements the aforementioned ER specification. The output of the execution of this application is depicted on the right of Fig. 2. The entities r^1 and r^2 of R give rise to the pair of entities (t^1, u^1) and (t^2, u^2) in the reconciled solution, respectively, as well as the relationships between them. On the other hand, the entity s^1 of S has already been reconciled in the related entities t^1 and u^1, while the entity s^2 derives the related entities t^3 and u^3. The value of the attribute D2 in the entities u^1 and u^3 is obtained through the function f5, whereas its value in the entity u^2 is taken from r^2.

Consider that the application has a defect in the projection from S to the reconciled solution, and an entity s^j is considered to be reconciled when f3(B1) is equal to the value of C1 in some entity of T or f4(B2) is equal to the value of D1 in some entity of U. If the application is not tested with meaningful data, the defect may not be detected and, as a result, the application could fail (for example, if the application is tested with the data of Fig. 2, the defect is not detected).

Due to the fact that it is crucial to ensure the correctness of the reconciliation process, it is essential to identify the important features to be tested, called *test conditions* [39]. From these test conditions, the situations of interest that are to be tested, called *test coverage items*, are derived by means of some adequacy criterion [40]. The test coverage items guide the generation of the test inputs of the test cases, and allow the tester to evaluate their adequacy. Regarding the testing of ER applications, the elaboration of these test inputs involves the state of the data sources before executing the reconciliation process (henceforth *test data sources*) and the information that constitutes a reconciled solution that is going to be updated during the reconciliation process (henceforth *test reconciled solution*).

For instance, one of the test conditions of the introductory example is "testing the generation of new entities and relationships in the reconciled solution from the table S." The test coverage items derived from this test condition that can detect the aforementioned defect are: 1) there is an entity s^j that corresponds to an entity t^k, but does not meet any related entity u^m; and 2) there is an entity s^j that corresponds to an entity u^m, but does not meet any related entity t^k. Fig. 3 shows the test inputs derived from these test coverage items (the number next to each node and each row of S indicates the test coverage item that is being covered).

Creating both the test data sources and the test reconciled solution is a crucial challenge, as the data stored are transformed to produce the test output and they have to contain enough meaningful data to adequately exercise the ER application.

The work presented in this paper deals with the definition of test models for integration testing, called *ITR Models*, which define the testing objectives (that is the test conditions) from the ER specification by means of a set of business rules called *integration rules*. These integration rules are specially focused on the subsequent derivation of test coverage items that guide the creation of the test data sources and the test reconciled solution.

The business rules, which are statements that define or constrain the business structure or the business behaviour [41], have been used in other approaches focused on testing database applications, such as [42] and [43]. On the other hand, as the integration rules are based on the system specification, they could also be used to generate some implementation of the ER application.

IV. Framework for Testing ER Applications

The framework for testing ER applications was proposed in our earlier work [10]. Fig. 4 depicts the architecture of the framework, which is composed of four main blocks.

1) *Data source models*: allow the representation of the information in the data sources that are to be reconciled, as well as the way of accessing them. These data sources can be a structured or an unstructured database, a web service, a warehouse, or other information generator.
2) *Reconciled solution model*: allows the software engineer, once data sources have been defined, to design the conceptual data model that represents the reconciled solution to be achieved, according to the ER problem domain, as a virtual graph.
3) *Transformations model*: represents the different transformations that the data in the sources must undergo in order to carry out the ER and to be consistent with the reconciled solution model. The description of this model is out of the scope of this work.
4) *Test models*: allow the representation of the testing objectives for the ER application in the early stages of the development (once the data sources and reconciled solution models have been defined). The test models can be fo-

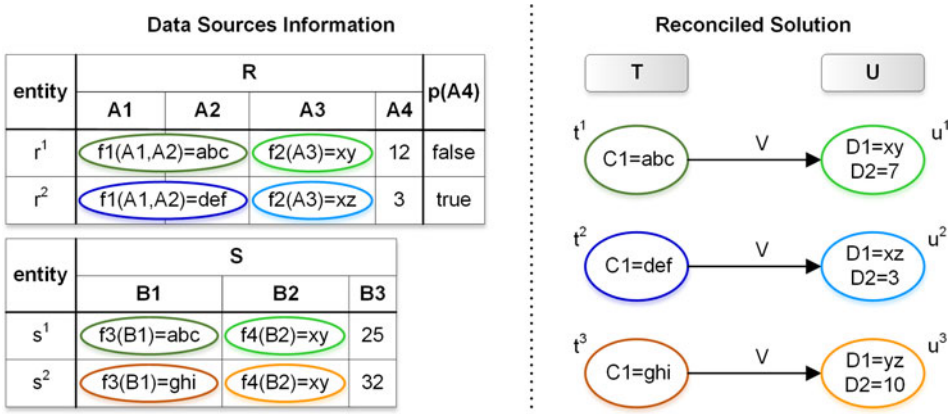

Fig. 2. Example of information stored in the data sources and the reconciled solution.

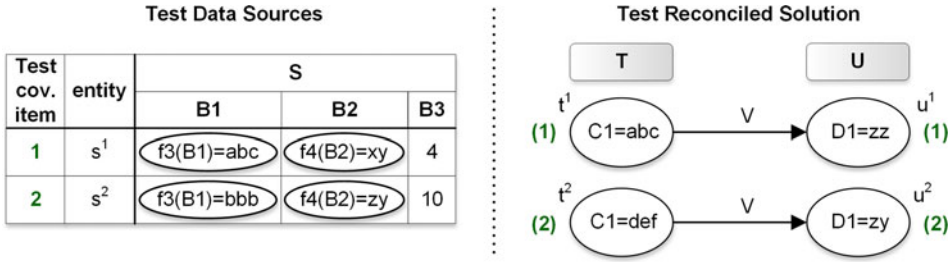

Fig. 3. Test inputs of the introductory example.

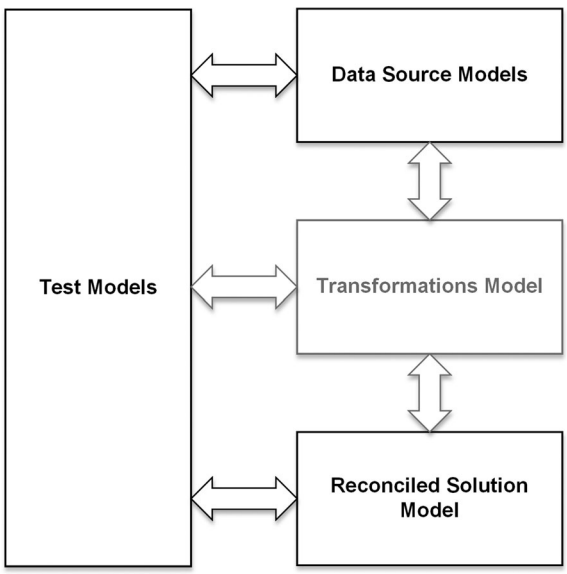

Fig. 4. Framework architecture.

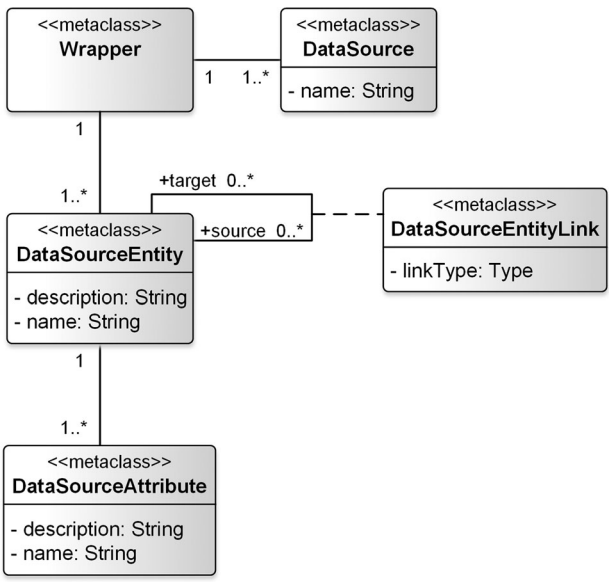

Fig. 5. Data source metamodel.

cused on different levels, such as unit testing, component testing or integration testing. This paper is focused on the definition of models for integration testing, called *ITR Models*, which are described in Sections IV-C and V.

The following sections describe the aforementioned models, which are representations of abstract models called metamodels. These metamodels provide all the elements that are necessary to create the models and include the attributes required to meet the standard ISO/IEC TR 24774 [44].

A. Data Source Models

Fig. 5 displays the metamodel that allows the creation of the data sources models. The metaclass *DataSource* represents each data source involved in the ER process. Data retrieved from each data source through the instantiation of the metaclass *Wrapper* will be structured in a set of types of entities (metaclass *DataSourceEntity*) that may be related to each other using the metaclass *DataSourceEntityLink*. These types of entities will be

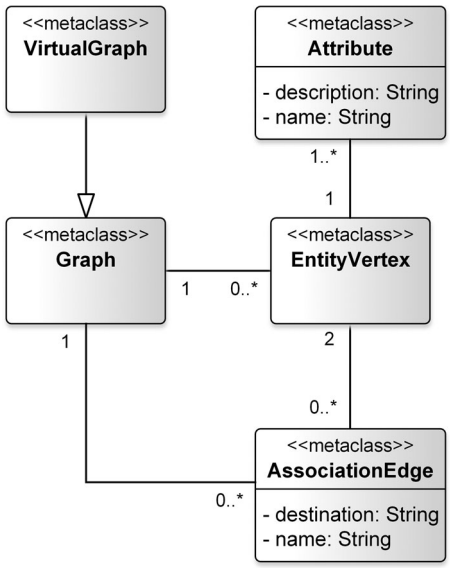

Fig. 6. Reconciled solution metamodel.

composed of a set of attributes (instantiation of the metaclass *DataSourceAttribute*) that describe the entities themselves.

B. Reconciled Solution Model

The next block of the framework is the reconciled solution model, which is based on a virtual graph. Graph technology is a natural solution to dealing with problems related to Big Data and especially for the relationships between entities. The wide variety of existing algorithms, for example Dijkstra, A$*$ or Kruskal among others, offer a great flexibility in providing solutions to different problems. Theoretically, graphs can be displayed in two ways: explicit and implicit. An explicit graph is a collection of items (vertices and edges) that can be completely stored in memory. An implicit (or virtual) graph is a graph that cannot be completely stored in memory for various reasons, such as size or hardware limitations [25].

With the implicit approach, it is possible to build structures on the fly. This will allow the building of different solutions to address many scenarios within a business logic where the predefined data model cannot meet the extensibility or availability of the required data sources. Considering the advantages that virtual graphs provide and the large amount of data that an ER process uses, this option has been the one selected for this proposal.

The elements that compose a reconciled solution model are shown in the metamodel of Fig. 6, which is an extended version of a graph metamodel [45]. It contains a set of vertices (metaclass *EntityVertex*) that represent the types of entities, which are composed of a set of attributes (metaclass *Attribute*). The vertices are related by a set of edges (metaclass *AssociationEdge*) that represent the types of relationships that can be established among entities. A *VirtualGraph* is modeled as an abstract class that implements the metaclass *Graph*.

Thus, the instantiation of the reconciled solution model is a virtual graph that stores the entities (and their relationships) that have been reconciled. The information stored in this virtual graph at a specific stage of the reconciliation process is called *current reconciled solution*, whereas the information stored after finishing the reconciliation process is called *final reconciled solution*.

C. ITR: a Test Model for Integration Testing

As stated above, this work is focused on the definition of test models for integration testing (called *ITR Models*), which are formed by a set of business rules, called *integration rules*, that represent the test conditions. The ITR model is created in the early stages of software development, taking into account the data source models, the reconciled solution model, and the ER specification stated by the expert.

Fig. 7 depicts the metamodel that represents the elements of the integration rules (represented by the metaclass *IntegrationRule*) that constitute the ITR.

1) *Integration context* (represented by the metaclass *IntegrationContext*) establishes the connections between the types of entities of one or several data source models and the types of entities of the reconciled solution model that are involved in a test condition, taking also into account the types of relationships among them. These types of entities and relationships are called *context entities* and *context relationships*, respectively. The connections impose conditions to be fulfilled in order to project the entities of the data sources to the entities of the reconciled solution. For instance, in the introductory example of Fig. 1 the IC of test condition 1 would relate R with T and U via the predicates f1(A1,A2) $==$ C1 and f2(A3) $==$ D1, as well as via the relationship V.
2) *Integration context view* or *view*, for short, (represented by the metaclass *IntegrationView*) connects a subset of the context entities involved in an integration context. A view is focused on a part of the projection defined by means of an integration context. For instance, a possible view of the integration context described above would relate R only with T, as the testing objective is focused on the projection between these two context entities.
3) *Integration pattern* (represented by the metaclass *IntegrationPattern*) imposes conditions on the context entities and context relationships involved in an integration context or view, as well as on their attributes, which are called *context attributes*. These conditions lead the actions of the ER process to be tested.

According to the integration pattern, our approach classifies the integration rules into *structural rules* (represented by the metaclass *Structural*) and *load rules* (represented by the metaclass *Load*). The structural rules impose conditions to be fulfilled in order to create new entities and relationships in the current reconciled solution. The load rules establish conditions to be fulfilled in order to derive the value of the attributes of the entities that belong to the current reconciled solution from the data sources. In addition, the load rules are classified into several types, according to two dimensions: the existence of preconditions (*conditional* and *nonconditional rules*) and the kind of condition to be fulfilled by the attributes (*IS*, *OR*, *AND*, and *XOR*

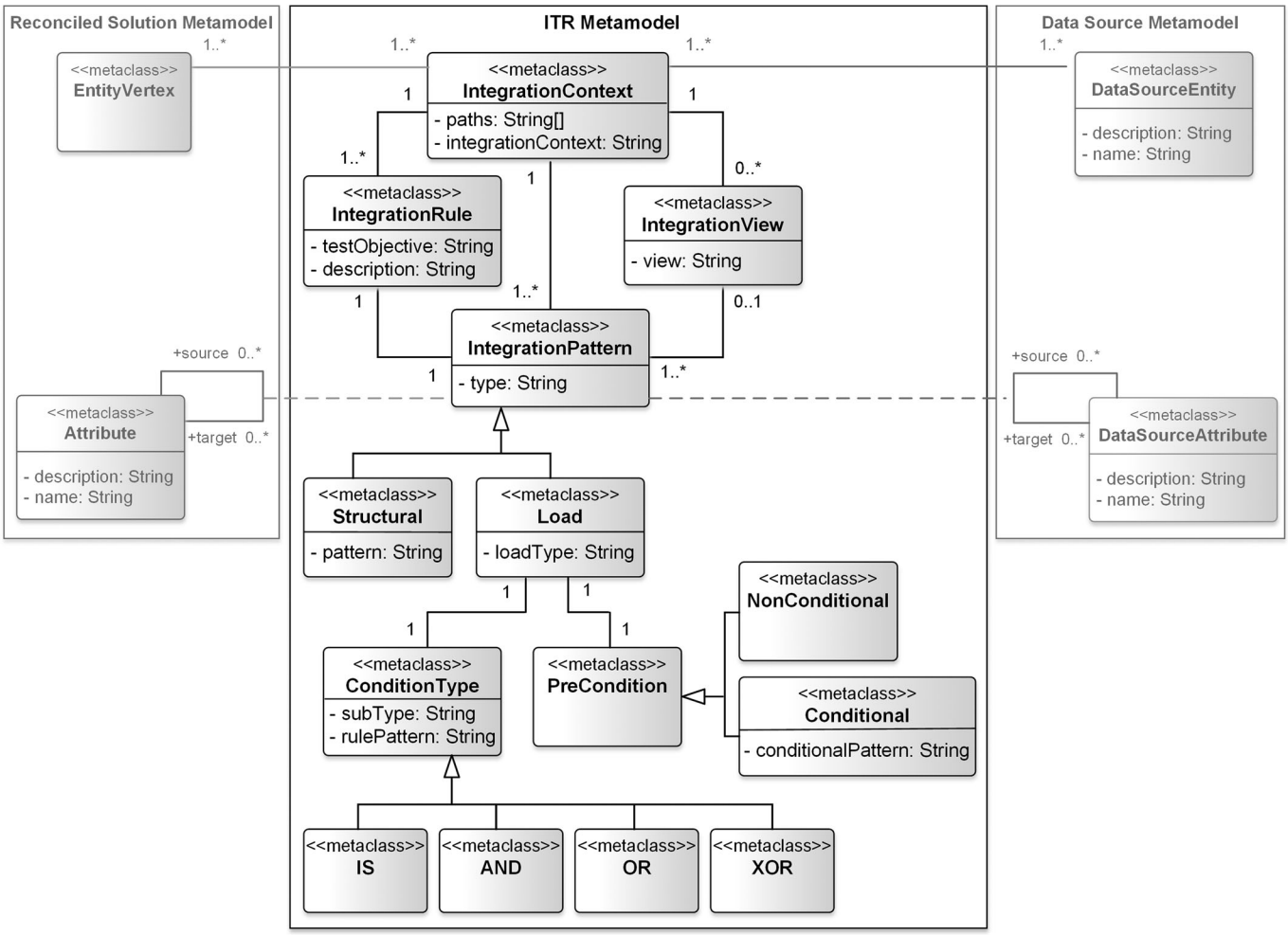

Fig. 7. ITR metamodel.

rules). The next section describes each type of integration rule and the language used in their construction.

V. Specification of Integration Rules

This section describes how the integration rules are constructed, using a language based on the SBVR specification [46] called RaQUEL (business Rules QUEry Language). The following sections present the patterns that allow the expression of the integration context, the integration context view, and the integration patterns of each type of integration rule, which are represented as attributes in the metaclasses of the ITR metamodel.

A. Specification of Integration Contexts and Integration Context Views

In order to describe the integration context and the integration context views of an integration rule, it is necessary to define the concept *path* that is used in their construction.

Definition 1: A *path* P is a set of one or more types of entities (instances of the metaclasses *DataSourceEntity* and *EntityVertex*) and/or types of relationships (instances of the metaclasses *AssociateEdge* and *DataSourceEntityLink*) $\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_n$, where each pair $(\mathrm{R}_i, \mathrm{R}_{i+1})$ is directly connected via some attributes in the predicate $\mathrm{q}_{i,i+1}$:

$$\text{Path P is } \mathrm{R}_1\ [\mathrm{q}_{1,2}]\ \mathrm{R}_2 [\mathrm{q}_{2,3}] \ldots [\mathrm{q}_{n-1,n}] \mathrm{R}_n .$$

Each $\mathrm{q}_{i,i+1}$ can contain arithmetic and logical expressions and functions, which involve attributes of $\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_{i+1}$.

Example 1: Consider the introductory example of Fig. 1, the path that relates the types of entities R, T, and U is defined as follows:

$$\text{Path P1 is R } [\mathrm{f1}(\mathrm{A1}, \mathrm{A2}) == \mathrm{C1}]\ \mathrm{T}\ [\mathrm{C1} == \text{source}]\ \mathrm{V}\ [\text{destination} == \mathrm{D1} \text{ and } \mathrm{f2}(\mathrm{A3}) == \mathrm{D1}]\ \mathrm{U}.$$

The definition of the concept *path* suggests the redefinition of the integration context and the integration context view, as well as the context entities, context relationships, and context attributes in terms of this concept, as explained next.

Definition 2: An *integration context* (IC) is a set of one or more paths $\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_m$ that define the connections between the data source models and the reconciled solution model that are involved in a test condition:

$$\text{Integration context IC is } \mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_m .$$

If an integration context is formed by only one path, it can be defined directly by

$$\text{Integration context IC is } \mathrm{R}_1 [\mathrm{q}_{1,2}]\ \mathrm{R}_2 [\mathrm{q}_{2,3}] \ldots [\mathrm{q}_{n-1,n}]\ \mathrm{R}_n .$$

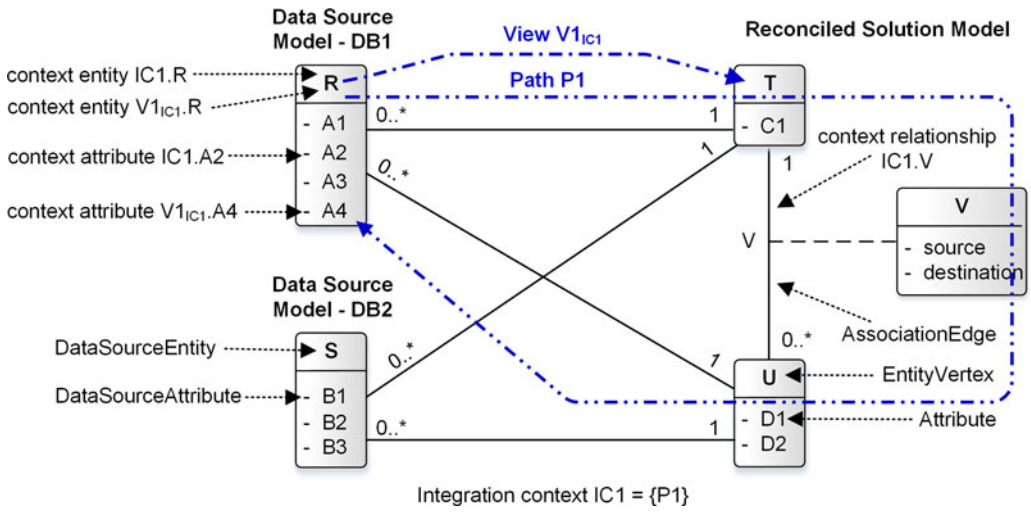

Fig. 8. Example of IC and IC view.

Example 2: Consider the introductory example of Fig. 1 and the path P1 defined in Example 1, the integration context of the test condition 1 is defined as follows:

Integration context IC1 is P1.

Definition 3: An *integration context view* or *view*, for short, (V_{IC}) of an integration context IC is a subset R_j, R_{j+1}, $R_{j+2}, \ldots, R_k$ of a path P of IC, where each pair (R_i, R_{i+1}) $(i = j \ldots k-1)$ is directly connected via the predicate defined in P:

Integration context view V_{IC} is $R_j[]R_{j+1}[]\ldots[]R_k$ of IC.P.

Example 3: Consider test condition 1 of the introductory example depicted in Fig. 1 and the integration context IC1 of Example 2. Consider that one of the testing objectives is focused on the projection from R to T. The view focused on this projection is defined as follows:

Integration context view $V1_{IC1}$ is R[]T of IC1.P1.

Definition 4: A *context entity* is a type of entity R of a path P of an integration context IC denoted by IC.R. If R is not unique in IC it is denoted by IC.P.R, where P is a path of IC that contains R. A context entity of a view V_{IC} of an integration context IC is denoted by V_{IC}.R.

Definition 5: A *context relationship* is a type of relationship R of a path P of an integration context IC denoted by IC.R. If R is not unique in IC it is denoted by IC.P.R, where P is a path of IC that contains R. A context relationship of a view V_{IC} of an integration context IC is denoted by V_{IC}.R.

Definition 6: A *context attribute* is an attribute A of a context entity or a context relationship of an integration context IC denoted by IC.A. If A is not unique in IC, it is denoted by IC.P.R.A or IC.R.A, where P is a path of IC and R is a context entity of P that contains A. A context attribute of a view V_{IC} of an integration context IC is denoted by V_{IC}.A or V_{IC}.R.A.

The set of paths involved in the definition of an integration context, as well as the specification of this integration context and its views are represented by the attributes *paths*, *integrationContext* and *view* of the ITR metamodel, respectively.

Fig. 8 depicts the connections between the data source models and the reconciled solution model of the introductory example, along with the previous concepts for the test condition 1. R and S (instances of *DataSourceEntity*) are connected to T and U (instances of *EntityVertex*) to represent the projections defined in the introductory example. The figure shows the path that relates R with T and U, which is used to define the integration context IC1 of test condition 1 and the view $V1_{IC1}$, as well as the context entities, the context relationship, and the context attributes of both IC1 and $V1_{IC1}$. Note that a type of entity (and also a type of relationship) can have several names, according to the different integration contexts and views in which they are involved. For example, the type of entity R is denoted by IC1.R, when it is involved in the integration context IC1, and $V1_{IC1}$.R, when it is involved in the view $V1_{IC1}$.

Once the data source models and the reconciled solution model have been instantiated, the integration contexts allow the identification of the data that have been reconciled, called *reconciled context domain*, and the data that have to be reconciled, called *unreconciled context domain*:

1) The *reconciled context domain* is composed of the instances of the context entities and context relationships that meet the conditions imposed by the paths that form this integration context.
2) The *unreconciled context domain* is composed of the rest of the instances that do not meet the conditions imposed by the paths of the integration context.

Similarly, the integration context views allow the identification of the reconciled and unreconciled data that are derived from the conditions they impose, called *reconciled view domain* and *unreconciled view domain*, respectively. These four data domains constitute the information on which the conditions imposed by the integration patterns of the integration rules are applied.

The following definitions indicate how the aforementioned data domains are obtained, taking into account definitions 1 to 6:

Definition 7: A *path domain* (D_P) of a path P is the data domain obtained from the Cartesian product of the instances of R_i $(i = 1 \ldots n)$ involved in P that fulfil the predicates $q_{i,i+1}$. Each element of a data domain is called from now on *tuple*.

Definition 8: A *reconciled context domain* (RD_{IC}) of an integration context IC is the data domain obtained from the Cartesian product of the tuples of the path domains D_{Pi} of the paths P_i $(i = 1 \ldots m)$ of IC that are equal on their common context attributes, along with the tuples of D_{Pi} that do not match any tuple.

Definition 9: An *unreconciled context domain* (UD_{IC}) of an integration context IC is the data domain formed by the instances of the context entities and context relationships of IC that are not included in the reconciled context domain RD_{IC} of IC.

Definition 10: A *reconciled view domain* (RD_V) of an integration context view V_{IC} is the data domain obtained from the Cartesian product of the instances of the subset R_j, $R_{j+1}, R_{j+2}, \ldots, R_k$ of the path P of IC involved in V_{IC}, which fulfil the predicates defined in P between each pair (R_i, R_{i+1}) $(i = j \ldots k\text{–}1)$.

Definition 11: An *unreconciled view domain* (UD_V) of an integration context view V_{IC} is the data domain composed of the instances of the context entities and context relationships involved in V_{IC} that are not included in the reconciled view domain RD_V of V_{IC}.

B. Specification of Structural Rules

A structural rule establishes the projection from a context entity IC.R (or $V_{IC}.R$) that belongs to a data source model to one or more context entities and context relationships $IC.S_i$ (or $V_{IC}.S_i$) that belong to the reconciled solution model. It also establishes one or several conditions on the context attributes $IC.S_i.A_j$ (or $V_{IC}.S_i.A_j$) that constrain their values when the new entities and relationships are added to the current reconciled solution.

The projection imposed by the structural rule must be fulfilled by each instance of IC.R (or V_{IC}.R) that belongs to the unreconciled context domain of IC (or the unreconciled view domain of V_{IC}). The integration pattern of a structural rule is described below, using the EBNF notation [47].

Definition 12: The integration pattern of a *structural rule* (represented by the attribute *pattern* of the metaclass *Structural* of the ITR metamodel) is defined as follows:

$$\begin{aligned}&\text{structural\_rule} = \text{“Each unreconciled”}\ (\text{IC.R}|\text{V}_{\text{IC}}.\text{R})\\ &\quad\text{“generates”}\,\text{gen\_cond}\{\text{“and”}\ \text{gen\_cond}\};\\ &\text{gen\_cond} = \text{“exactly one”}\ (\text{IC.S}_i|\text{V}_{\text{IC}}.\text{S}_i)\text{“with”}\ \text{att\_cond};\\ &\text{att\_cond} = (\text{IC.S}_i.\text{A}_j|\text{V}_{\text{IC}}.\text{S}_i.\text{A}_j)\text{“ = ”}\,\text{p}_j\{\text{“and”}\\ &\quad(\text{IC.S}_i.\text{A}_j|\text{V}_{\text{IC}}.\text{S}_i.\text{A}_j)\text{“ = ”}\,\text{p}_j\}\end{aligned}$$

where each p_j is a predicate over context attributes of IC.R (or V_{IC}.R) and/or $IC.S_i$ (or $V_{IC}.S_i$).

Example 4: Consider the introductory example of Fig. 1 and the view described in Example 3, the integration pattern of the structural rule that imposes conditions to project R to T is defined as follows:

$$\begin{aligned}&\text{Each unreconciled V1}_{\text{IC1.}}\text{R generates exactly one V1}_{\text{IC1.}}\text{T}\\ &\qquad\text{with}\,\text{V1}_{\text{IC1.}}\text{T.C1} = \text{f1(A1, A2)}.\end{aligned}$$

This structural rule establishes that each instance of R that belongs to the unreconciled view domain of $V1_{IC1}$ generates a new instance of T in the current reconciled solution. It also indicates that the value of the attribute C1 of this new instance of T must be the result of the function f1 over the attributes A1 and A2 of R.

C. Specification of Load Rules

A load rule imposes one or more conditions that constrain the value of a context attribute IC.S.A that belongs to the reconciled solution model, according to one or several context attributes $IC.R_i.B_j$ that belong to the data source models. The conditions must be fulfilled by each tuple of the reconciled context domain of IC.

As stated in this section, the load rules are classified according to two dimensions. The first dimension indicates whether a load rule establishes preconditions that have to be fulfilled before constraining the value of a context attribute (*conditional rules*), or it does not establish any precondition (*nonconditional rules*).

The second dimension indicates the types of conditions that constrain the value of the context attributes according to one or several predicates (IS, OR, AND, XOR *rules*). These predicates can be either arithmetical or logical expressions or functions over context attributes of the IC, as well as constants or context attributes of IC. The evaluation of the predicates returns a value that fits the type of the context attribute constrained or a *null* value, which indicates that the predicate was not able to reach a concrete value (for example, because of a context attribute used in a function does not exist, or because a context attribute has a unknown value in its data source).

The following definitions describe the patterns of each category, using the EBNF notation.

Definition 13: A *conditional rule* is a load rule whose integration pattern is defined as follows:

$$\text{conditional\_rule} = \text{“If”}\ p\ \text{“then”}\ \text{rule\_pattern};$$

where p is a predicate over context attributes IC.S.A and/or $IC.R_i.B_j$ whose evaluation returns a Boolean value. This predicate defines the preconditions to be fulfilled before constraining the value of the context attribute IC.S.A by means of *rule_pattern*. This pattern is written according to Definitions 14–17 described next.

Definition 14: An IS *rule* is a load rule that constrains the value of a context attribute IC.S.A, such that it must be equal to the evaluation of a predicate *p*. The integration pattern is defined as follows:

$$\text{IS\_rule} = \text{“Each”}\ \text{IC.S.A}\ \text{“is”}\ p;$$

Definition 15: An AND *rule* is a load rule that constrains the value of a context attribute IC.S.A, such that it must be formed by the union of the evaluations of the predicates p_i that do not return a *null* value. The integration pattern is defined as follows:

$$\begin{aligned}&\text{AND\_rule} = \text{“It is obligatory that”}\ \text{IC.S.A}\\ &\qquad\text{“is composed of”}\ p_i\{\text{“and”}\ p_i\};\end{aligned}$$

Definition 16: An OR *rule* is a load rule that constrains the value of a context attribute IC.S.A, such that it can be formed by the evaluation of one or several predicates p_i that do not return

a *null* value. The integration pattern is defined as follows:

$$\text{OR\_rule} = \text{“It is permitted that” IC.S.A “is composed of” } p_i \{\text{“or” } p_i\};$$

Definition 17: An *XOR rule* is a load rule that constrains the value of a context attribute IC.S.A, such that it must be equal to the evaluation of only one predicate p_i. Each predicate p_i has a different priority n_i (n_i = 1, 2, etc., where 1 is the highest priority) that indicates the order in which they are evaluated. IC.S.A takes the value of the first predicate p_i that does not return a null value. The integration pattern is defined as follows:

$$\text{XOR\_rule} = \text{prioritization “Each” IC.S.A “is only” } p_i\{\text{“or”}p_i\};$$
$$\text{prioritization} = p_i \text{ “has priority” } n_i\{p_i\text{“has priority” } n_i\};$$

The integration patterns of nonconditional IS, nonconditional AND, nonconditional OR, and nonconditional XOR rules (or IS, AND, OR, and XOR rules, for short) are directly described by Definitions 14–17, respectively. These integration patterns are represented by the attribute *rulePattern* of the metaclass *ConditionType* of the ITR metamodel. In contrast, the conditional IS, conditional AND, conditional OR, and conditional XOR rules are defined by combining Definition 13 with Definitions 14–17, respectively. The integration pattern of these conditional rules is represented by the combination of the attributes *conditionalPattern* of the metaclass *Conditional* and *rulePattern* of the metaclass *ConditionType* of the ITR metamodel.

Example 5: Consider the introductory example of Fig. 1. The conditional IS rules that represent the reconciliation of the attribute D2 are defined as follows:

1) Path P1 is R [f1(A1,A2) = = C1] T [C1 = = source] V [destination = = D1 and f2(A3) = = D1] U.
2) Path P2 is S [f3(B1) = = C1] T [C1 = = source] V [destination = = D1 and f4(B2) = = D1] U.
3) Integration context IC is P1, P2.
4) If p(IC.A4) then IC.D2 is IC.A4.
5) If ! p(IC.A4) then D2 is f5(IC.A4, IC.B3).

Statements 1–3 define the integration context IC, whereas statements 4 and 5 define the integration pattern of two conditional IS rules. Note that the integration context IC can be used to define several integration rules.

After defining the test conditions as a set of the integration rules, the test coverage items can be derived by means of applying logic criteria [48], [49] over the conditions imposed by these integration rules. This process is illustrated in the next section through a case study.

VI. Case Study

To evaluate the proposed framework and the ITR model for integration testing, two real-word problems have been used as case studies. The first case study makes use of the specification of an application called DIPHDA (Dynamic Integration for Patrimonila Heritage Data in Andalucía) that aims to reconcile historical heritage data of Andalusia (Spain). The second case study involves a real-word problem that is being studied by the University of Seville pertaining to reconciling the digital information related to the research publications of its researchers and collaborators, known as the REPORTS project (Reconciling rEsearch PrOjects infoRmation and publicaTions for the university of Seville).

The DIPHDA application is also used in this section to illustrate how the ITR model can be created and how it can be used to derive the test coverage items from the test conditions. The following sections present both the DIPHDA application and the REPORTS project, describe the ITR models created, and provide a summary of the test coverage items that were derived. Finally, a discussion of the approach is presented.

A. Case Study 1: DIPHDA Application

The management of historical and cultural heritage information in Andalusia (Spain) is being addressed by the cultural council of the region using a horizontal and global system called “MOSAICO” [50]. The aims of this system are: 1) to offer a global information system that stores information about the historical and cultural heritage of the region; 2) to offer technological resources and tools for the management of this information; and 3) to bring the general public and Government more specific (and relevant) information related to historical heritage. This system was developed to meet the objectives of the cultural council, such as managing, protecting, preserving, and promulgating the cultural heritage of Andalusia, as well as bringing government services to the citizens of the region.

The information related to the numerous cultural and historical monuments of Andalusia is stored in several data sources, and therefore it is very difficult to control all information published about historical heritage in a global context. In addition, the size and complexity of these data sources make the management of these systems complicated due to the large amount of information stored on them. It is therefore necessary to reconcile the existing information about monuments from all available data sources.

Considering this problem, the DIPHDA application is being developed with the collaboration of the Fujitsu Laboratories of Europe. The objective of DIPHDA is to achieve significantly improved accuracy and data management efficiency, based on reconciliation logic applied to open data information, as opposed to simple string matching reconciliation. This solution will be capable of integrating different data sources. For this particular case, the data sources “MOSAICO,” Wikipedia, and Yelp are going to be used.

Our approach aims to generate the ITR model while DIPHDA is being developed, so that it can be used not only to guide the testing of the application but also to verify the requirements of the ER problem. Due to the ITR model is composed of a set of business rules written in a language based on SBVR, the ER expert can easily understand it, with the result that missing requirements may be discovered or inconsistent requirements may be detected.

Fig. 9 depicts the data source models and the reconciled solution model of DIPHDA. The classes *Mosaico*, *DBPedia,* and *Yelp* (instances of the metaclass *DataSourceEntity*) model the

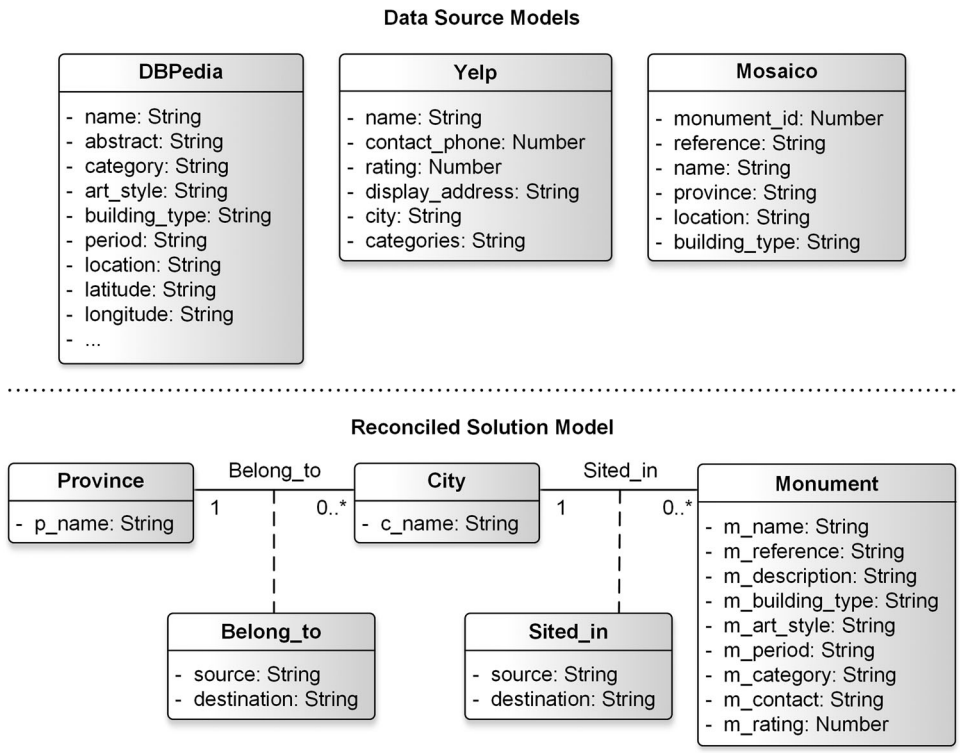

Fig. 9. Data source models and reconciled solution model of DIPHDA.

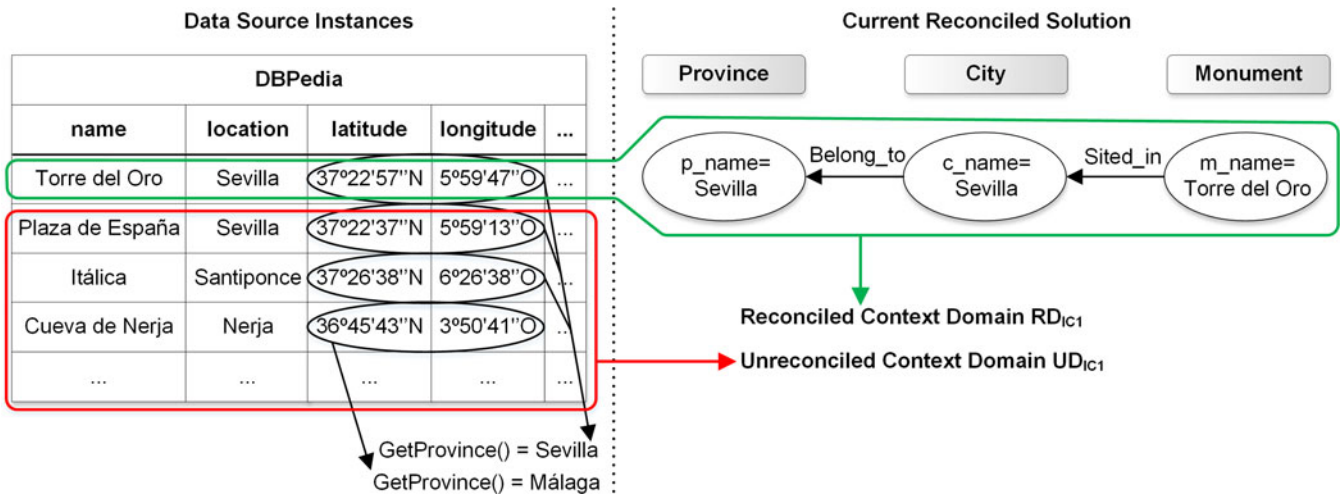

Fig. 10. Example of several instances of DBPedia and a current reconciled solution.

historical heritage elements of the data sources MOSAICO, Wikipedia, and Yelp, respectively, that the cultural council of Andalusia is going to use to perform the ER. The reconciled solution model is composed of the classes *Monument*, *City,* and *Province* (instances of the metaclass *EntityVertex*) that represent the types of entities considered necessary to carry out the reconciliation of the historical heritage elements, as well as the association classes *Belong_to* and *Sited_in* (instances of the metaclass *AssociationEdge*), which represent the relationships between these types of entities. The attribute *p_name* of the reconciled solution model must be unique for each instance of *Province*, while the attributes of the other classes are not constrained by the unique restriction due to the possibility of two cities that belong to different provinces having the same name, or two monuments with the same name being sited in two different cities.

Fig. 10 shows an example of the instances of *DBPedia*, as well as a current reconciled solution. The instances of *DBPedia* are rows of a table stored in a database, and each row represents a historical heritage element to be reconciled. Similarly, the instances of *Mosaico* and *Yelp* are rows of a table of a database too. On the other hand, the instances of *Monument*, *City,* and *Province* are the nodes of the virtual graph that represent the entities stored in the current reconciled solution, whereas the

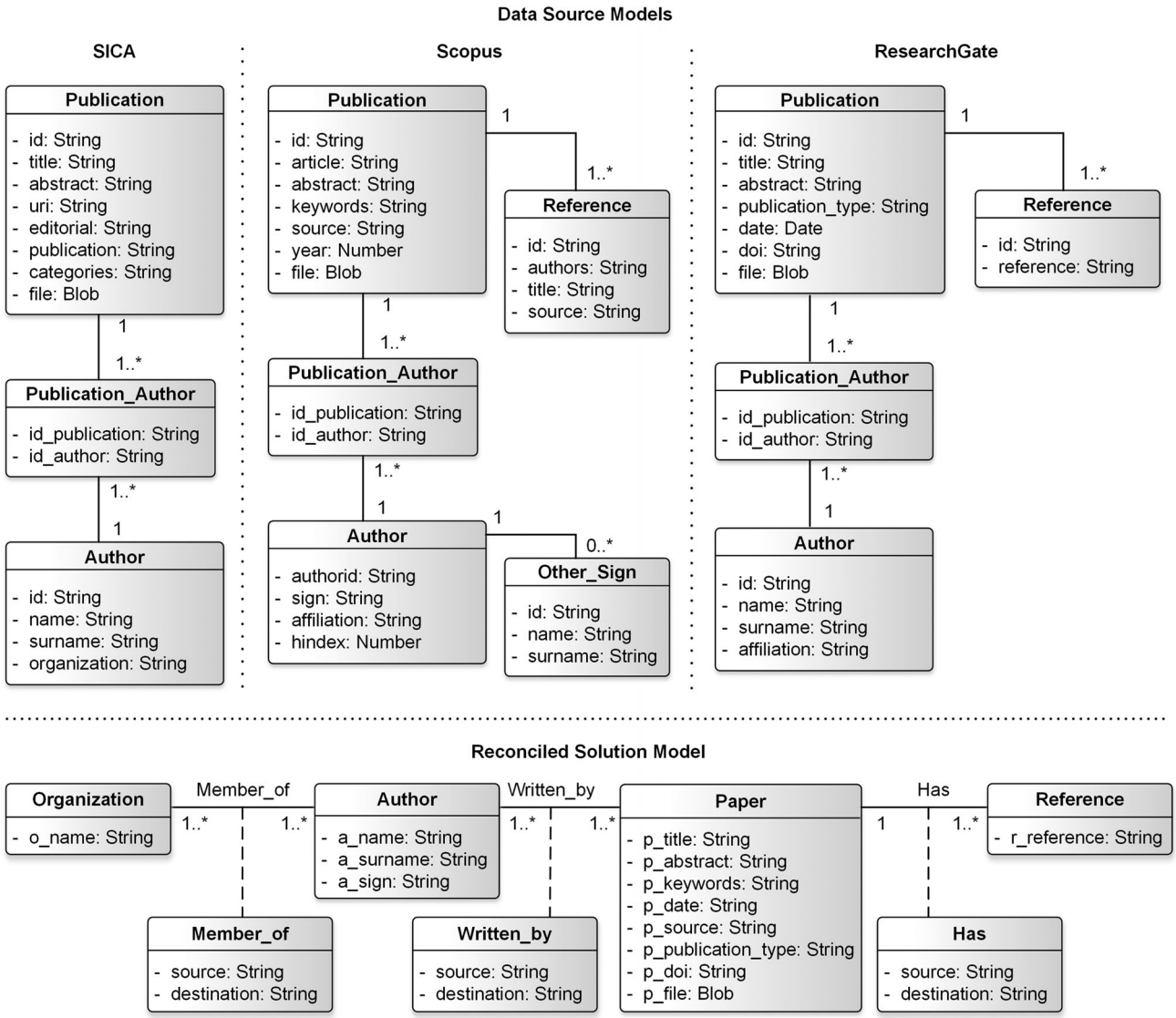

Fig. 11. Data source models and reconciled solution model of the REPORTS project.

instances of *Belong_to* and *Sited_in* are the edges between nodes (that is, the relationships between entities).

B. Case Study 2: REPORTS Project

The information related to the research activities and the results of the researchers at the University of Seville, such as research projects and publications, can be found in many different resources: proprietary databases like SISIUS (the institutional repository for community members at the University of Seville), SICA (the institutional repository for researchers of the Andalusian region in Spain), the data sources of other universities, abstract and citation databases, and social networking sites like ResearchGate, etc. The reconciliation of this information is an important issue to address, not only to maintain knowledge about the research activities of the university and community but also to report correct information to the research community.

Researchers at the University of Seville have to report their research activities and results to be disseminated and evaluated by different institutional authorities. These research activities and results are included and managed in sources like SICA [53], SISIUS [54], or even in ResearchGate by researchers and institutional authorities. In lot of cases, this information has many inconsistencies, most frequently due to the mistakes made by researchers and institutional authorities when information is managed and, in some cases, due to other factors including the maintainability of different systems, among others. For instance, a new version of the SICA system was developed and an important migration was performed in recent years. The consequences of this migration were that a lot of information was affected. In addition, most publications are automatically indexed in international systems like Scopus, Springer Link, Web of Science, etc. As a result, it is a very complex task for researchers and institutional authorities to control, manage, and evaluate all of this information. The REPORTS projects aims to help researchers

Integration context:

(1) Path P1 is DBPedia
[Equals(GetProvince(latitude, longitude), p_name)] Province
[Belong_to.destination=p_name] Belong_to
[Belong_to.source=c_name and Equals(location, c_name)] City
[Sited_in.destination=c_name] Sited_in
[Sited_in.source=m_name and Equals(name, m_name)] Monument

(2) Integration context IC1 is P1

Integration context views:

(3) Integration context view V1 is DBPedia [] Province of IC1.P1

(4) Integration context view V2 is DBPedia [] Province [] Belong_to [] City of IC1.P1

Integration patterns:

(5) Each unreconciled V1.DBPedia generates exactly one V1.Province
with V1.p_name= GetProvince(V1.latitude, V1.longitude)

(6) Each unreconciled V2.DBPedia generates exactly one V2.City with V2.c_name=V2.location
and exactly one V2.Belong_to with V2.source=V2.c_name and V2.destination=V2.p_name

(7) Each unreconciled IC1.DBPedia generates exactly one IC1.Monument with IC1.m_name=IC1.name
and exactly one IC1.Sited_in with IC1.Sited_in.source=IC1.m_name and
IC1.Sited_in.destination=IC1.c_name

Fig. 12. Structural rules to project DBPedia to the reconciled solution.

and institutional authorities to reduce efforts and improve the information quality for the dissemination and evaluation of their research activities and results, by means of the ER of several data sources.

As with the case study of the DIPHDA application, this case study is focused on the early testing of the REPORTS project, while it is still under study. The case study considers three data sources; a proprietary store (SICA), Scopus, and ResearchGate. Fig. 11 depicts the data source models and the reconciled solution model. The classes *Organization*, *Author*, *Paper,* and *Reference* (instances of the metaclass *EntityVertex*) represent the type of entities involved in the reconciliation process, which are related by the associations *Member_of*, *Written_by*, and *Has* (instances of the metaclass *AssocciationEdge*).

C. ITR Models

The ITR models of DIPHDA and the REPORTS project were designed from their ER specifications, considering both the data sources and the reconciled solution models. The ITR model of DIPHDA is composed of 18 integration rules: 9 structural rules, 2 conditional IS rules, 1 conditional OR rule, 3 IS rules, 1 AND rule, and 2 XOR rules. The ITR model of the REPORTS project is formed by 21 integration rules: 11 structural rules, 3 conditional XOR rules, 2 IS rules, 1 OR rule, 1 AND rule, and 3 XOR rules. First, we designed the structural rules that lead the creation of new entities and relationships in the current reconciled solution. After that, we designed the different types of load rules that constrain the value of the attributes. To illustrate how the structural and load rules are created, the next sections present the details regarding the DIPHDA application.

1) Structural Rules of the DIPHDA Application: According to the ER specification, each historical heritage element stored in MOSAICO, Wikipedia, and Yelp is represented in the reconciled solution by means of an entity *Monument* sited in an entity *City* that belongs to an entity *Province*. The specification also indicates the attributes and functions that lead the projection from the data sources to the reconciled solution. For example, Fig. 12 shows the statements of the three structural rules designed to project *DBPedia* to the reconciled solution.

Statements 1 and 2 specify the integration context IC1 formed by the path P1, which relates *DBPedia* with *Province*, *City,* and *Monument*. This integration context is shared by the three structural rules. The order of the connections between the context entities was established according to the cardinalities one-to-many of the reconciled solution model. The predicates of P1 impose the conditions to be fulfilled to reconcile the instances of the aforementioned context entities, and they usually involve a similarity function called *Equals*. This function determines whether two strings can be considered equal, according to a specific degree of similarity. For example, to determine whether an instance of *DBPedia* corresponds to some instance of *Province*, the evaluation of the function *GetProvince* (which returns the name of a province from the angular distances represented by the attributes *latitude* and *longitude* of *DBPedia*) must be *equal* to the attribute *p_name* of *Province*.

The integration context IC1 allows the identification of the data that have been reconciled from the data source Wikipedia (represented by the context entity *DBPedia*), that is, the reconciled context domain RD_{IC1}, and the data that have not been reconciled yet, that is, the unreconciled context domain UD_{IC1}. Fig. 10 shows both data domains.

Statements 3 and 4 of Fig. 12 define the integration context views V1 and V2, which are focused on the projection from *DBPedia* to *Province* and *City*, respectively. Note that V2 relates *DBPedia* with *City* via *Province*, due to the one-to-many cardinality between *Province* and *City*. V1 and V2 are subsets of the path P1 that relax the conditions imposed by IC1 in or-

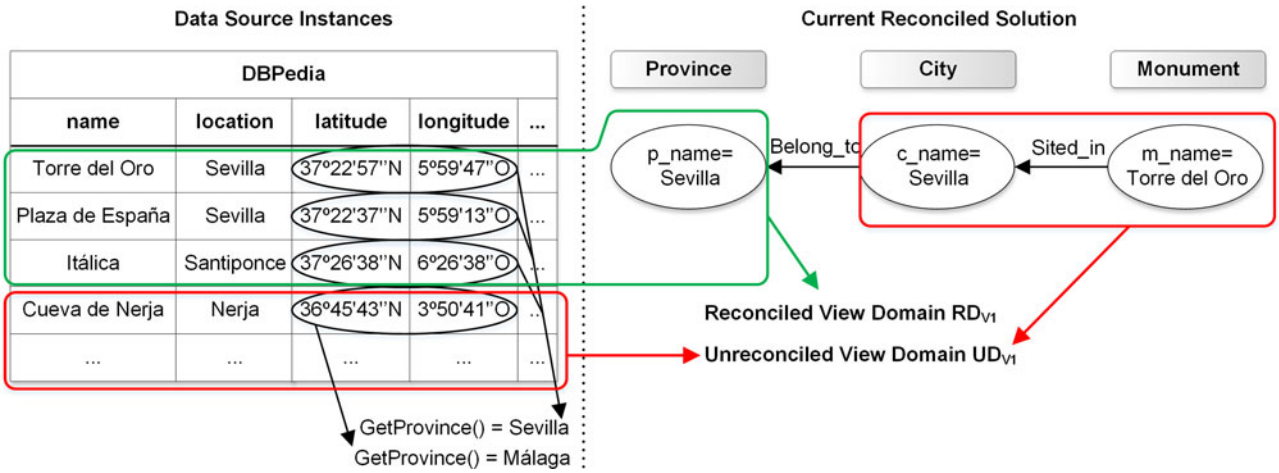

Fig. 13. Example of the reconciled view domain and unreconciled view domain of the view V1.

Integration context:

(1) Path P1 is DBPedia
[Equals(GetProvince(latitude, longitude), p_name)] Province
[Belong_to.destination=p_name] Belong_to
[Belong_to.source=c_name and Equals(location, c_name)] City
[Sited_in.destination=c_name] Sited_in
[Sited_in.source=m_name and Equals(name, m_name)] Monument

(2) Path P2 is Yelp
[Equals(GetProvince(GetZipCode(display_address)), p_name)] Province
[Belong_to.destination=p_name] Belong_to
[Belong_to.source=c_name and Equals(city, c_name)] City
[Sited_in.destination=c_name] Sited_in
[Sited_in.source=m_name and Equals(name, m_name)] Monument

(3) Integration context IC2 is P1, P2

Integration Pattern:

(4) IC2.DBPedia.building_type has priority 1

(5) IC2.Yelp.categories has priority 2

(6) Each IC2.Monument.m_building_type is only IC2.building_type or IC2.categories

Fig. 14. XOR rule that reconciles the attribute m_building_type of Monument from DBPedia and Yelp.

der to obtain the unreconciled view domains that contain the instances of both *Province* and *City* to be reconciled. Fig. 13 depicts the reconciled view domain RD_{V1} and the unreconciled view domain UD_{V1} of the view V1.

Statements 5, 6, and 7 of Fig. 12 define the integration patterns of the three structural rules that state the conditions that lead the creation of new entities and relationships into the current reconciled solution: instances of *Province* (statement 5), *City* and *Belong_to* (statement 6), as well as *Monument* and *Sited_in* (statement 7). The order of these statements constrains the order in which the entities and relationship have to be created, according to the cardinalities one-to-many established in the reconciled solution model. Thus, the instances of *Province* should be created before the instances of *City*, which should be created before the instances of *Monument*.

For example, statement 5 establishes that each instance of *DBPedia* that belongs to the unreconciled view domain of V1 (see UD_{V1} in Fig. 13) generates a new instance of *Province*. Therefore, DIPHDA should generate the node *Province* "Málaga." On the other hand, statement 6 indicates that each instance of *DBPedia* included in the unreconciled view data derived of V2 generates a new instance of *City* and *Belong_to*. As a result, DIPHDA should create the nodes *City* "Santiponce" and "Nerja," as well as two relationships *Belong_to*: one relationship between "Santiponce" and "Sevilla" and another one between "Nerja" and "Málaga."

2) Load Rules of the DIPHDA Application: The ER specification establishes several requirements to derive the value of the attributes of the new entities stored in the reconciled solution from the aforementioned data sources. Thus, nine load rules were designed. Fig. 14 displays one of these load rules: an XOR rule that reconciles the value of the context attribute *IC2.Monument.m_building_type* from attributes of the context entities *DBPedia* and *Yelp*.

TABLE I
NUMBER OF INTEGRATION RULES AND TEST COVERAGE ITEMS OF CASE STUDY 1 (DIPHDA APPLICATION)

Type of integration rule	Number of rules	Number of test coverage items
Structural	9	108
Conditional IS	2	53
Conditional OR	1	30
IS	3	60
AND	1	21
XOR	2	52
Total:	18	324

TABLE II
NUMBER OF INTEGRATION RULES AND TEST COVERAGE ITEMS OF CASE STUDY 2 (REPORTS PROJECT)

Type of integration rule	Number of rules	Number of test coverage items
Structural	11	183
Conditional XOR	3	115
IS	1	44
OR	2	35
AND	1	35
XOR	3	144
Total:	21	556

Statements 1–3 define the integration context *IC2* composed of the paths *P1* and *P2*, which relate *DBPedia* and *Yelp* with *Province*, *City,* and *Monument*, as explained in the section above. Statements 4–6 define the integration pattern of the XOR rule that imposes the conditions to be fulfilled to derive the value of the context attribute *IC2.Monument.m_building_type*. Statements 4 and 5 specify the prioritization of the context attributes *IC2.DBPedia.building_type* and *IC2.Yelp.categories,* whereas statement 6 establishes that *IC2.Monument.m_building_type* can only obtain its value from one of these context attributes. As a result, for each tuple that belongs to the reconciled context domain of *IC2*, first *IC2.DBPedia.building_type* is evaluated. If this evaluation does not return a *null* value, *IC2.Monument.m_building_type* takes this value. Otherwise, it takes the value of the evaluation of *IC2.Yelp.categories*. Note that if both evaluations of *IC2.DBPedia.building_type* and *IC2.Yelp.categories* return a *null* value, the context attribute *IC2.Monument.m_building_type* also has an unknown value.

D. Test Coverage Items

After defining the ITR models, we applied a Masking modified condition decision coverage (MCDC)-based criterion over the conditions imposed by the integration rules to derive the test coverage items, that is, the situations of interest to be tested. This criterion has demonstrated its utility in previous work, such as [51] (for testing SQL queries) and [42] (for testing the user–database interaction).

The masking MCDC criterion requires that every condition in a logical decision has taken on all possible outcomes at least once, every decision has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect the decision's outcome [52]. In our case, each integration rule gives rise to a logical decision, formed by the conditions imposed by the integration context (or the integration context view) and the integration pattern.

To automatically obtain the test coverage items, we used the SQLFpcWS web service [51], which implements the masking MCDC criterion. Tables I and II show the number of test coverage items derived from each type of integration rule. After that, we generated the test inputs, which are composed of the test data sources and the test reconciled solution that cover the test coverage items, as a part of our early testing strategy. Afterwards, we automatically evaluated the coverage achieved.

In order to generate the test inputs for the first case study (DIPHDA application), our first approach began from the populated data sources MOSAICO, Wikipedia, and Yelp, which accumulated 26 632 rows. Despite the large number of rows, the percentage of coverage achieved was about 2%. Since comparison between actual and expected outputs becomes more difficult with large test databases, we began from empty test data sources, in order to keep them small and meaningful. The same approach was taken for the REPORTS project, and we began from empty test data sources.

Tables III and IV display the number of instances inserted into the test data sources and the reconciled solution to achieve total

TABLE III
NUMBER OF INSTANCES OF THE TEST DATA SOURCES AND TEST RECONCILED SOLUTION OF CASE STUDY 1 (DIPHDA APPLICATION)

	Type of entity or relationship	Number of instances
Test data sources	Mosaico	48
	DBPedia	77
	Yelp	40
Test reconciled solution	Province	69
	City	65
	Belong_to	75
	Monument	58
	Sited_in	61
	Total:	493

TABLE IV
NUMBER OF INSTANCES OF THE TEST DATA SOURCES AND TEST RECONCILED SOLUTION OF CASE STUDY 2 (REPORTS PROJECT)

	Type of entity or relationship	Number of instances
Test data source SICA	Publication	21
	Publication_Author	27
	Author	65
Test data source Scopus	Publication	22
	Publication_Author	29
	Author	64
	Other_Sign	59
	Reference	11
Test data source	Publication	21
ResearchGate	Publication_Author	26
	Author	64
	Reference	10
Test reconciled solution	Organization	6
	Author	63
	Member_of	71
	Paper	22
	Written_by	27
	Reference	14
	Has	11
	Total:	633

coverage for both case studies. All 324 coverage items derived for the DIPHDA application and all 556 coverage items derived for the REPORTS project are covered when evaluated over 493 and 633 instances of test data, respectively.

E. Fault Detection

Achieving a high test coverage is essential in order to test the functionality of the application thoroughly, with the aim of improving the quality of the final software product. Furthermore, developing test cases for increasing the coverage will also increase the fault detection ability of the test cases [51]. The above results show that the information stored in the data sources of the DIPHDA application (26 632 rows) covers a low number of test coverage items (about 2%), with the result that most of the meaningful situations to be tested are not exercised and therefore, a fairly large number of possible defects are not detected.

In contrast, by covering all the test coverage items derived from the ITR models of both case studies (using 493 rows for the DIPHDA application and 633 rows for the REPORTS project), the following types of defects may be detected.

1) Faults in the projection from the context entities of the data source models to the reconciled solution model. These faults may produce failures in the creation of new instances in the current reconciled solution, as well as during the derivation of the value of the attributes that belong to the instances that form the current reconciled solution.
2) Faults in the implementation of the ER specification that guide the reconciliation of the context attributes of the instances stored in the current reconciled solution, causing failures when their values are derived.
3) Faults owing to the incorrect management of *null* values or missing information. These faults may cause failures when the instances of the data sources are projected to the current reconciled solutions and when the attribute values are derived.

To illustrate how we can detect the aforementioned faults and failures, consider the following test coverage items derived for the DIPHDA application. The test coverage item 1 is derived from the structural rule whose integration pattern is depicted in statement 7 of Fig. 12, whereas the test coverage items 2, 3, and 4 are derived from the XOR rule of Fig. 14.

1) *Test coverage item 1*: there is an instance of *DBPedia* that meets an instance of *City* and an instance of *Monument*, which are related by an instance of *Sited_in*, but it does not meet any instance of *Province*.
2) *Test coverage item 2*: there is an instance of *DBPedia* (d^i) that meets a set of instances *Province* (p^i), *City* (c^i) and *Monument* (m^i), which are related by instances of *Belong_to* and *Sited_in*. There is an instance of *Yelp* (y^j) that meets a set of instances *Province* (p^j), *City* (c^j), and *Monument* (m^j), which are related by instances of *Belong_to* and *Sited_in*. Besides, $m^i.m\_name$ is equal to $m^j.m\_name$, $c^i.c\_name$ is different to $c^j.c\_name$, $p^i.p\_name$ is equal to $p^j.p\_name$, and $d^i.building\_type$ is different to $y^j.categories$.
3) *Test coverage item 3*: there is an instance of *DBPedia* (d^i) that meets a set of instances *Province* (p^i), *City* (c^i), and *Monument* (m^i), which are related by instances of *Belong_to* and *Sited_in*. There is an instance of *Yelp* (y^j) that meets the same instances p^i, c^i, and m^i. Besides, $d^i.building\_type$ is different to $y^j.categories$.
4) *Test coverage item 4*: there is an instance of *DBPedia* (d^i) that meets a set of instances *Province* (p^i), *City* (c^i), and *Monument* (m^i), which are related by instances of *Belong_to* and *Sited_in*. There is an instance of *Yelp* (y^j) that meets the same instances p^i, c^i and m^i. Besides, the evaluation of $d^i.building\_type$ returns a *null* value and the evaluation of $y^j.categories$ does not return a *null* value.

Fig. 15 depicts an example of some test data sources and a test reconciled solution that covers the foregoing test coverage items. The number on the left of each row of the test data sources and the numbers in brackets below each node and relationship of the test reconciled solution indicate the test coverage items that need these instances so that they can be covered. For example, to cover the test coverage item 1, row 1 of *DBPedia* that does not meet any *Province* node, along with the related nodes *City* with *c_name* $=$ *c1* and *Monument* with *m_name* $=$ *m1* are needed. Note that the test reconciled solution represents the current reconciled solution at the initial stage of the reconciliation process to be tested, which is going to be updated during the execution of the test cases.

The expected output of the execution of the test cases that use the previous test data sources and test reconciled solution, according to the structural rules of Fig. 12 and the XOR rule of Fig. 14, is shown in Fig. 16. This expected output is formed by the state that the final reconciled solution should have after the execution of the test cases and it is called *expected reconciled solution*. DIPHDA should generate the three new nodes highlighted in the figure, along with their relationships, from row 1 of *DBPedia*. It should also derive the value of the attribute *m_building_type* of the *Monument* nodes (highlighted in the figure) from the other rows of *DBPedia* and *Yelp*.

Next, we illustrate that generating test inputs to exercise the test coverage items allows a more thorough testing that is able to detect a number of defects in the implementation. Consider a faulty implementation of DIPHDA that transforms the test inputs of Fig. 15 into the final reconciled solution of Fig. 17. This final reconciled solution, which is the observed output of the execution of the test cases and is called *observed reconciled solution*, reveals the existence of several defects.

1) Fault 1: the implementation does not check the cardinality between *Province* and *City*, which was specified in the reconciled solution model (that is, it has a defect in the management of the reconciled solution model during the projection of the entities). This defect produces failure 1 of Fig. 17: the node *City* with *c_name* $=$ *c1* is connected with two different *Province* nodes. As a result, the observed reconciled solution does not conform to the reconciled solution model.
2) Fault 2: the decision that checks whether a row of *DBPedia* corresponds to a set of related nodes *Province*, *City,* and *Monument* of the current reconciled solution is not correct,

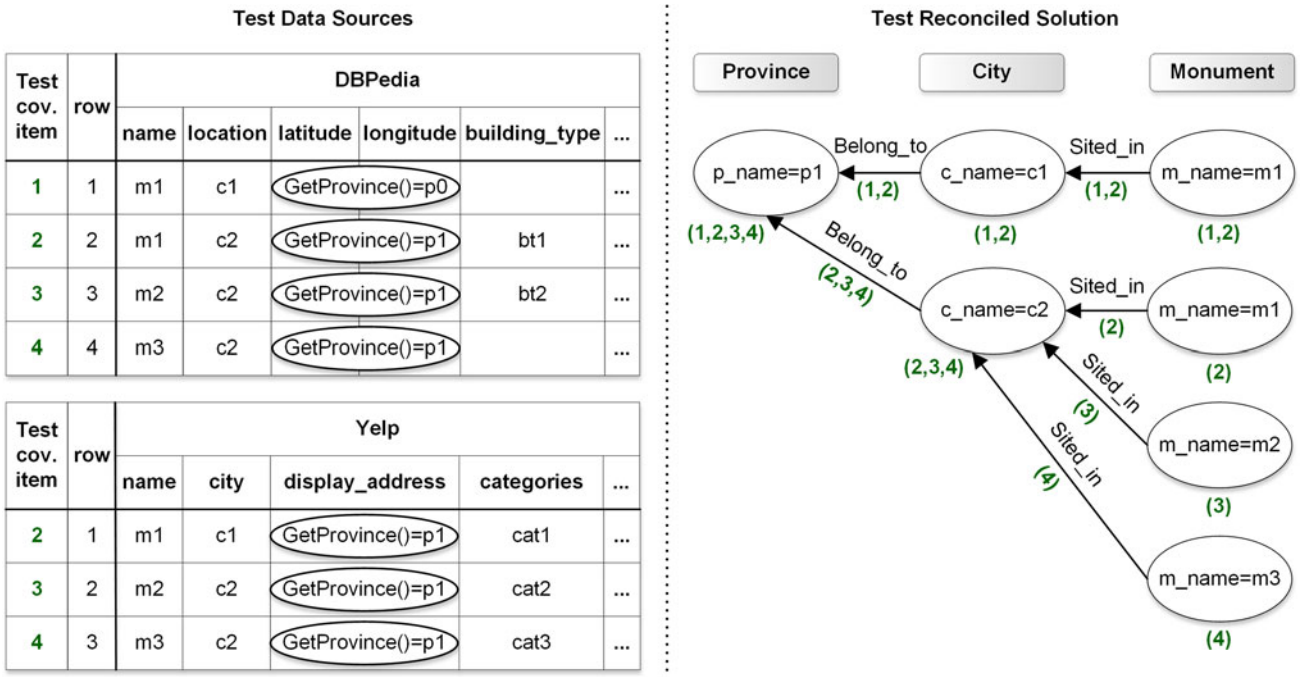

Fig. 15. Example of test data sources and test reconciled solution.

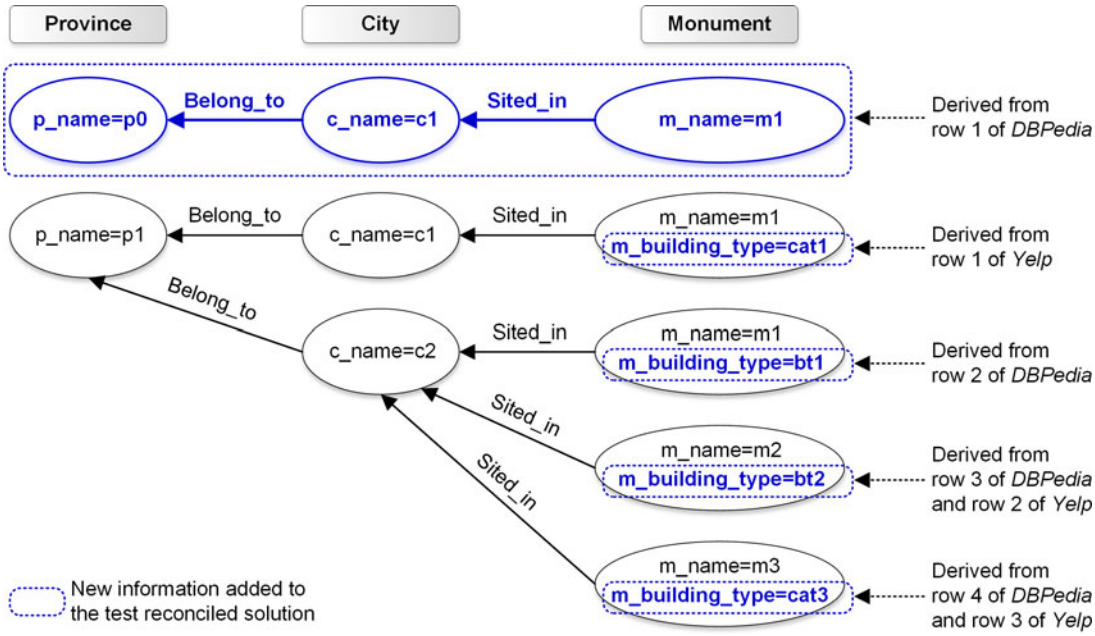

Fig. 16. Expected reconciled solution.

because it only checks the nodes *Province* and *Monument* (that is, the implementation has a defect in the projection from *DBPedia*). This defect causes DIPHDA to consider that row 2 of *DBPedia* corresponds to the set of related nodes *Province* with $p\_name = p1$, *City* with $c\_name = c1$, and *Monument* with $m\_name = m1$, which is the same set of related nodes that corresponds to row 1 of *Yelp*. As a result, when the context attribute *m_building_type* is derived through the XOR rule of Fig. 14, it has the value "bt1" instead of "cat1" (see failure 2 of Fig. 17).

3) Fault 3: the implementation considers that the context attribute *categories* has the higher priority, instead of *building_type*, when the context attribute *m_building_type* is derived (that is, it has a defect in the prioritization of the context attributes involved in the XOR rule of Fig. 14). This defect produces failure 3 of Fig. 17: the context attribute *m_building_type* of the *Monument* node with $m\_name = m2$ has the value "cat2," instead of "bt2."
4) Fault 4: the implementation does not include a decision to check whether the context attribute *building_type* of

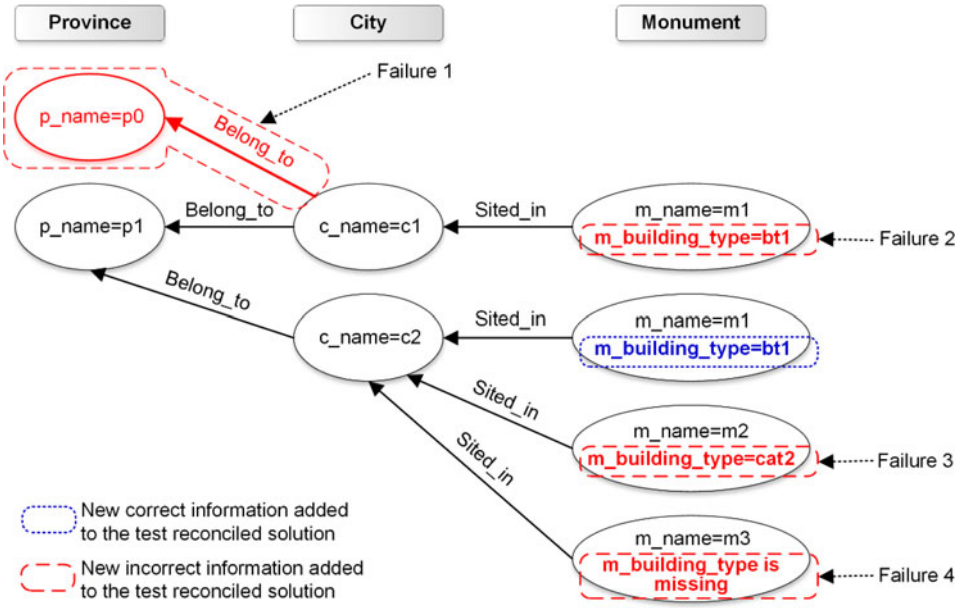

Fig. 17. Observed reconciled solution.

DBPedia has a missing value when it is used to derive the value of *m_building_type* through the XOR rule of Fig. 14 (that is, it has a defect in the management of missing information). This defect produces failure 4 of Fig. 17: the attribute *m_building_type* is missing in the *Monument* node with *m_name* = *m3*, instead of having the value "cat3."

Because the information stored in the 26 632 rows included in the data sources does not cover the test coverage items 1, 2, and 4, faults 1, 2, and 4 would not be detected. However, we were able to detect these defects by designing the tests to cover the aforementioned test coverage items.

F. Discussion

The results of the case studies show that we derived a set of test coverage items from the ITR model to guide the generation of meaningful test inputs of test cases that are able to detect defects in the ER application. These test coverage items were derived systematically from the ITR model and represent interesting situations that are easy to forget when an application is being developed. Besides, the number of instances in the test data sources and the test reconciled solution that cover the test coverage items of the DIPHDA application is considerably lower than the number of rows of the production data sources, and the coverage is considerably higher than that achieved using these production data sources. Therefore, it is possible to thoroughly test the functional suitability of these types of applications with a small amount of data. An additional advantage is that we reduce the effort of designing the expected output and comparing it against the actual output. However, there are several issues that may limit our approach, which are discussed below.

First, the ITR metamodel may not provide all the elements that are necessary to describe the whole class of ITR models, that is, the models that represent the testing objectives of every ER domain. To address this issue, different ER domains should be analyzed to determine whether the current ITR metamodel has to be extended with new metaclasses, attributes, and relationships, so that it allows the creation of ITR models for these domains. The extension of the ITR metamodel could also lead to the extension and/or the adaptation of other related metamodels of the framework for testing ER applications.

Second, the integration rules that constitute the ITR models may not be expressive enough to represent all of the important features that are to be tested of the ER application. To mitigate this limitation, the integration rules can be extended in order to include more complex conditions and deal with comparisons based on functions in detail. Besides, there are some logical formulations of SBVR that have not been considered in the definition of the integration rules yet, which can be used to extend their expressiveness.

Finally, the study is limited to the early testing of the real application DIPHDA and the REPORTS project, obtaining similar results on both. Therefore, the real effectiveness of the test cases that were designed in the early stages of the development has not yet been validated. This validation is going to be carried out when the applications of the case studies are available. However, the results of the case studies demonstrate that we have obtained test coverage items that guide the generation of test cases which can detect defects that could be present in the implementation.

VII. Conclusion and Future Work

This work presents an integration of early testing to an ER application. The previous work presented in [10] has been further developed, giving rise to the ITR model. This model is based on four main pillars: the reconciled solution model (that represents the solution to be achieved), the data sources models (that represent to data sources to reconcile), the transformations

model (that represents the transformation that data must undergo in the different stages of the ER process), and the test models (that represent the testing objectives for the ER). Also, testing objectives have been represented as business rules in order to automatically derive the test coverage items by the application of the MCDC criterion.

The main contributions of this work may be summarized as: 1) the description of the elements that constitute the framework for testing the ER applications; 2) the definition of the ITR model for integration testing, which represents the testing objectives as a set of business rules, called integration rules; and 3) the application of the proposal to two real-world problems.

This approach has been validated with two real-world case studies based on the heritage information management of the region of Andalusia (Spain) and the publications of the researchers of the University of Seville. After applying the test coverage items derived from the ITR models, it was found that there are three main type of faults: those related to the projection from the context entities of the data source models to the reconciled solution model, those related to the implementation of the ER specification that guide the reconciliation of the attributes of the instances stored into the current reconciled solution, and those related to the incorrect management of *null* values or missing information.

It has been verified that the addition of early integration testing to the ER application is capable of detecting a series of deficiencies, which *a priori* were not known and that will help to improve the final result that the ER application offers. Furthermore, applying early testing with a test model that allows the use of the test coverage to guide the test case design process has made it possible to reduce the amount of data that need to be stored in the data sources used for testing, thereby achieving a more exhaustive testing that covers 100% of the test coverage items.

Future work encompasses several avenues such as the definition of the transformations that automate the process of generating the test cases, the extension of the test metamodel to cover the unit testing of the transformations applied over the data to carry out the ER (represented by the transformation model) and the identification of different case studies to validate the approach.

REFERENCES

[1] C. C. Yang, H. Chen, and K. Hong, "Visualization of large category map for Internet browsing," *Decis. Support Syst.*, vol. 35, no. 1, pp. 89–102, 2003.

[2] J. G. Enríquez, F. J. Domínguez-Mayo, M. J. Escalona, J. A. García-García, V. Lee, and G. Masatomo, "Entity identity reconciliation based big data federation—A MDE approach," in *Proc. 2015 Int. Conf. Inf. Syst. Develop.*, 2015.

[3] H. Wang, *Innovative Techniques and Applications of Entity Resolution*. Hershey, PA, USA: IGI Global, 2014.

[4] L. Getoor and A. Machanavajjhala, "Entity resolution: Theory, practice & open challenges," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2018–2019, 2012.

[5] F. Wang, H. Wang, J. Li, and H. Gao, "Graph-based reference table construction to facilitate entity matching," *J. Syst. Softw.*, vol. 86, no. 6, pp. 1679–1688, 2013.

[6] A. Gal, "Uncertain entity resolution: Re-evaluating entity resolution in the big data era: Tutorial," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1711–1712, Aug. 2014.

[7] L. Getoor and A. Machanavajjhala, "Entity resolution for big data," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 1527–1527.

[8] D. C. Schmidt, "Guest Editor's introduction: Model-driven engineering," *IEEE Comput.*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[9] J. Bézivin, "On the unification power of models," *Softw. Syst. Model.*, vol. 4, no. 2, pp. 171–188, 2005.

[10] J. G. Enríquez, R. Blanco, F. J. Domínguez-Mayo, J. Tuya, and M. J. Escalona, "Towards an MDE-based approach to test entity reconciliation applications," in *Proc. 7th Int. Workshop Autom. Test Case Des. Sel. Eval.*, 2016, pp. 74–77.

[11] I. Bhattacharya and L. Getoor, "A latent dirichlet allocation model for entity resolution," in *Proc. 2005 SIAM Int. Conf. Data Mining*, 2005, pp. 47–58.

[12] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita, "Declarative data cleaning: Language, model, and algorithms," in *Proc. 27th Int. Conf. Very Large Data Bases*, Jul. 2001, pp. 371–380.

[13] H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky, "Deterministic coreference resolution based on entity-centric, precision-ranked rules," *Comput. Linguist.*, vol. 39, no. 4, pp. 885–916, 2013.

[14] V. S. Verykios, G. V. Moustakides, and M. G. Elfeky, "A Bayesian decision model for cost optimal record matching," *VLDB J.*, vol. 12, no. 1, pp. 28–40, 2003.

[15] N. Vesdapunt, K. Bellare, and N. Dalvi, "Crowdsourcing algorithms for entity resolution," *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1071–1082, 2014.

[16] T. Williams and M. Scheutz, "POWER: A domain-independent algorithm for probabilistic, open-world entity resolution," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 1230–1235.

[17] W. E. Winkler, "Methods for record linkage and Bayesian networks," Technical Report, Statistical Research Division, US Census Bureau, 2002.

[18] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 269–278.

[19] W. W. Cohen and J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 475–480.

[20] J. Feng, J. Wang, T. Kraska, and M. J. Franklin, "CrowdER: Crowdsourcing entity resolution," *Proc. VLDB Endow.*, vol. 5, pp. 1483–1494, Jul. 2012.

[21] J. Fisher, P. Christen, and Q. Wang, "Active learning based entity resolution using Markow logic," *Adv. Knowl. Discovery Data Mining*, vol. 5476, pp. 338–349, 2016.

[22] E. Ioannou, W. Nejdl, C. Niedere'e, Y. Velegrakis, and C. Nieder, "On-the-fly entity-aware query processing in the presence of linkage," *Proc. VLDB Endow.*, vol. 3, no. 1, pp. 429–438, 2010.

[23] J. Zhao, P. Wang, and K. Huang, "A semi-supervised approach for author disambiguation in KDD CUP 2013," in *Proc. 2013 KDD Cup 2013 Workshop*, 2013, pp. 1–8.

[24] H. Wang, J. Li, and H. Gao, "Efficient entity resolution based on subgraph cohesion," *Knowl. Inf. Syst.*, vol. 46, no. 2, pp. 285–314, Feb. 2016.

[25] J. Mondal and A. Deshpande, "Managing large dynamic graphs efficiently," in *Proc. 2012 Int. Conf. Manage. Data*, 2012, pp. 145–156.

[26] P. Malhotra, P. Agarwal, and G. Shroff, "Graph-parallel entity resolution using LSH and IMM," in *Proc. CEUR Workshop*, vol. 1133, pp. 41–49, 2014.

[27] P. Vassiliadis, "A survey of Extract–Transform–Load technology," *Int. J. Data Warehousing Mining*, vol. 5, no. 3, pp. 1–27, 2009.

[28] J. Singh and K. Singh, "Statistically analyzing the impact of automated ETL testing on the data quality of a data warehouse," *Int. J. Comput. Elect. Eng.*, vol. 1, no. 4, pp. 488–495, 2009.

[29] N. ElGamal, A. El Bastawissy, and G. Galal-Edeen, "Towards a data warehouse testing framework," in *Proc. 2011 9th Int. Conf. ICT Knowl. Eng.*, 2012, pp. 65–71.

[30] S. B. Dakrory, T. M. Mahmoud, and A. A. Ali, "Automated ETL testing on the data quality of a data warehouse," *Int. J. Comput. Appl.*, vol. 131, no. 16, pp. 9–16, 2015.

[31] D. G. Tesfagiorgish and L. JunYi, "Big data transformation testing based on data reverse engineering," in *Proc. 2015 IEEE 12th Int. Conf. Ubiquitous Intell. Comput., 2015 IEEE 12th Int. Conf. Auton. Trusted Comput., 2015 IEEE 15th Int. Conf. Scalable Comput. Commun. Its Assoc. Workshops*, 2015, pp. 649–652.

[32] N. Li, A. Escalona, Y. Guo, and J. Offutt, "A scalable big data test framework," in *Proc. 2015 IEEE 8th Int. Conf. Softw. Testing Verification Validation*, 2015, pp. 1–2.

[33] J. Gutiérrez, G. Aragón, M. Mejías, F. Jose, D. Mayo, and C. M. R. Cutilla, "Automatic test case generation from functional requirements in NDT," in *Proc. Int. Conf. Web Eng.*, 2012, pp. 176–185.

[34] P. André, J.-M. Mottu, and G. Sunyé, "COSTOTest: A tool for building and running test harness for service-based component models (demo)," in *Proc. 2016 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 437–440.

[35] S. Nogueira, A. Sampaio, and A. Mota, "Test generation from state based use case models," *Formal Asp. Comput.*, vol. 26, no. 3, pp. 441–490, 2014.

[36] B. P. Lamancha, M. Polo, D. Caivano, M. Piattini, and G. Visaggio, "Automated generation of test oracles using a model-driven approach," *Inf. Softw. Technol.*, vol. 55, no. 2, pp. 301–319, 2013.

[37] J. J. Gutiérrez, M. J. Escalona, and M. Mejías, "A model-driven approach for functional test case generation," *J. Syst. Softw.*, vol. 109, pp. 214–228, 2015.

[38] A. A. Sofokleous and A. S. Andreou, "Automatic, evolutionary test data generation for dynamic software testing," *J. Syst. Softw.*, vol. 81, no. 11, pp. 1883–1898, 2008.

[39] *29119-1:2013—ISO/IEC/IEEE International Standard for Software and Systems Engineering—Software Testing—Part 1: Concepts and Definitions*, ISO/IEC/IEEE 29119-1:2013(E), vol. 2013, pp. 1–64, 2013.

[40] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, 1997.

[41] D. Hay and K. A. Healy, "Defining business rules: what are they really?," Final Report, Revision 1.3, The Business Rules Group, 2000.

[42] R. Blanco, J. Tuya, and R. V. Seco, "Test adequacy evaluation for the user-database interaction: A specification-based approach," in *Proc. IEEE 2012 5th Int. Conf. Softw. Testing Verification Validation*, 2012, pp. 71–80.

[43] D. Willmor and S. M. Embury, "Testing the implementation of business rules using intensional database tests," in *Proc. 2006 Testing Acad. Ind. Conf. Pract. Res. Techn.*, 2006, pp. 115–126.

[44] *Systems and Software Engineering—Life Cycle Management—Guidelines for Process Description*, ISO/IEC TR 247742010, 2010.

[45] S. Mazanek, "HelloWorld! An instructive case for the transformation tool contest," in *Proc. 2011 5th Transformation Tool Contest*, Zurich, Switzerland, Jun. 29–30, 2011, vol. 74, pp. 22–26.

[46] *Sematics of Business Vocabulary and Business Rules, Version 1.4, OMG Document Number: formal/2017-05-05*, OMG, vol. 2017, 2017.

[47] *Information Technology—Syntactic Metalanguage—Extended BNF*, Int. Stand. 149771996(E), vol. 1996, pp. 1–24, 1996.

[48] G. Kaminski, G. Williams, and P. Ammann, "Reconciling perspectives of software logic testing," *Softw. Testing Verification Rel.*, vol. 18, no. 3, pp. 149–188, 2008.

[49] G. Kaminski, P. Ammann, and J. Offutt, "Improving logic-based testing," *J. Syst. Softw.*, vol. 86, no. 8, pp. 2002–2012, 2013.

[50] *MOSAICO: Sistema de Información Para la gestión del Patrimonio Cultural en Andalucía*, MOSAICO. [Online]. Available: http://www.juntadeandalucia.es/cultura/web/areas/bbcc/sites/consejeria/areas/bbcc/contenidos/Mosaico/sistema_gestion_bienes_culturales. Accessed: Dec. 2017.

[51] J. Tuya, M. J. Suárez-Cabal, and C. De La Riva, "Full predicate coverage for testing SQL database queries," *Softw. Testing Verification Rel.*, vol. 20, no. 3, pp. 237–288, 2010.

[52] J. J. Chilenski, "An investigation of three forms of the modified condition decision coverage (MCDC) criterion," Security, Apr. 2001.

[53] *SICA: Scientific System Information of Andalusian*, SICA. [Online]. Available: https://sica2.cica.es/. Accessed: Dec. 2017.

[54] *SISIUS: Information System about Researching at the University of Seville*, SISIUS. [Online]. Available: https://investigacion.us.es/sisius. Accessed: Dec. 2017.