Modeling Ontology Semantic Constraints in Relational Database Management System

Aman Achpal, Vinayshekhar Bannihatti Kumar and Kavi Mahesh

***Abstract*—The Web Ontology Language, OWL, provides an effective language for encoding semantic constraints on data. Although a variety of semantic functionalities such as inference, retrieval, data integration, consistency and validation are enabled by representing semantics explicitly in OWL, most data continues to be managed in relational databases where such constraints can only be handled in procedural code buried in the application. This paper shows how the most important semantic constructs in OWL can be readily implemented in a database by semi-automatically translating OWL statements into database triggers without loss of encoded semantic information.**

***Index Terms*—Constrains, OWL, ontology, RDBMS, semantics, trigger**

I. Introduction

A data model is intended to be a representation of real-world entities, relations and their semantics [1]. The traditional relational database schema defines the structure of the data in terms of relations, attributes, keys and dependencies. Other aspects of the semantics of the real-world domain are handled by object-oriented models with their class hierarchies and inheritance mechanisms. However, representation of more complex semantic constraints continues to be a challenge in traditional databases. On the other hand, languages like the W3C recommendation of Web Ontology Language, OWL [2] are designed specifically to support constructs for capturing a rich set of semantic constraints on the real-world data [3]. The methods of the Semantic Web amount to more modern, comprehensive, self-sufficient and machine interpretable forms of data description. Yet, current implementations of OWL are best considered prototypes and are not easily scalable beyond a few million triples, to the sizes of data handled by a typical database.

Implementing a sufficiently rich set of semantic constraints in a relational database involves the tedious task of manually hard-coding the semantic constraints in the application programs. This causes further inefficiencies down the road when the ontology of the domain needs a change and the semantic constraints need to be edited. For example, consider a semantic constraint that says every customer must have at least one bank account with at least a minimum balance as prescribed by that bank. Such a constraint can only be implemented by hard-coding its rules into application programs or embedding a rather messy piece of code in suitable triggers in the database. In an OWL ontology, such constraints can be encoded rather neatly.

There is a need for a system or framework which encapsulates the fluidity and flexibility of describing the semantic constraints present in real-world entities and relationships, which at the same time can scale to capture the amount of data present in real-world applications. Furthermore, in-order to be employed at scale, this system must cater to individuals with training in legacy relational database management systems, in order to be a practical and viable replacement, while at the same time, proving to be a self-contained system capable of reasoning and factual interpretation, on the lines of the Semantic Web.

This paper proposes a solution to this problem by showing how an ontology can be semi-automatically mapped to SQL commands that can be run and stored in any commercial or open-source Relational Database Management System. The proposed approach implements the semantics of any domain with minimal effort and without loss of semantics. The semantics of the domain can be independently modeled using OWL tools such as OntoEdit, Protégé [4], or TopBraid Composer [5].

II. Current Approaches

There has been ample work in migrating Relational Databases into the Semantic Web. Some notable examples are the work done by Astrova et al. [6], Guntar Bumans [7], Alalwan et al. [8], Shen et al. [9] and RONTO [10]. There is, however, little work done in the opposite direction, which is, adding the semantics of Semantic Web technologies to a database, with a few exceptions [11]. One technique is to store the additional semantic information as meta-data in a separate layer on top of the relational database. This is the approach taken by GEM [12] and the work done by Vyšniauskas et al. in their hybrid approach [13]. The solution proposed in this paper is distinctly different, as it maps the semantic constraints directly to the database during the creation of the relational schema itself. At run-time, triggers whose code is automatically generated are used to apply the semantic constraints. A somewhat related solution is the rule-based approach in the work done by Astrova et al. [14]. However, their system is only able to store information that is supported inherently by the

Manuscript received December 08, 2015; revised January 20, 2016. This work is supported in part by the World Bank/Government of India research grant under the TEQIP programme (subcomponent 1.2.1) to the Centre for Knowledge Analytics and Ontological Engineering (KAnOE), http://kanoe.org at PES University, Bangalore, India.

Aman Achpal is an undergraduate student in Computer Science at PES Institute of Technology, Bangalore, India (e-mail: aman.achpal@gmail.com).

Vinayshekhar Bannihatti Kumar is an undergraduate student in Computer Science at PES Institute of Technology, Bangalore, India (e-mail: vinayshekhar000@gmail.com).

Dr. Kavi Mahesh is the Dean of Research, Director of KAnOE and Professor of Computer Science at PES University (phone: +91 9845290073 e-mail: drkavimahesh@gmail.com).

constructs of an RDMS, with no mechanism to create custom triggers corresponding to semantic constraints.

Some work has also gone into trigger based model, a notable example of which is by A. Tzacheva et al. [15] where they discuss using integrity constraints as a means of enforcing Domain-Range constraints, and triggers as a means of enforcing Subclass-Superclass constraints. The work, however, neither deals with the specific constructs of OWL, nor provides a framework or mapping for them. Finally, SQOWL2 [16] varies from our work in two main ways: in the underlying design philosophy of the system, SQOWL2 implements constraints such as a Functional Property by hard-coding them within the same database table. In the proposed solution, the mapping is done in an open ended manner to obtain the same consistency through automatically generated triggers regardless of the ontology, without relying on exploiting the structure of the database to enforce the constraints. The second difference lies in the range of semantic constructs from OWL supported in the proposed solution including Cardinality constraints, AllValuesFrom, SomeValuesFrom, and additional specific constructs that are known to be important and useful in modeling the semantics of real world systems.

In what follows, we describe the mappings from OWL semantic constructs to database triggers in sufficient detail to ensure reproducibility and easy implementation of this work in any database system whose semantics is defined as an OWL ontology.

III. Mapping Structural Constraints: OWLToRDBMS

Fig. 1 shows the overall architecture of the mapping from OWL to RDBMS. The various elements of the OWL ontology that need to be mapped to the relational system are:

- Classes (Specified as IRI or URI declarations)
- Properties
  - Datatype Properties
  - Object Properties
- Restrictions on Classes and Properties
- Other OWL constructs and Individuals (i.e., instances)

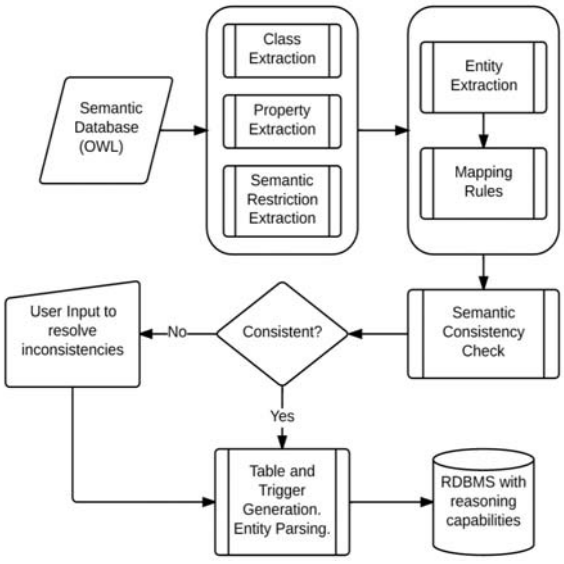

Fig. 1. Architecture of OWL to RDBMS Mapping

In order to map them, the following steps are taken:

i. Pellet (or any other OWL 2 DL) reasoner is used to parse the OWL ontology and build the hierarchy of classes in the ontology.
ii. The properties of the ontology are iterated over, and the hierarchy of properties is built.
iii. These hierarchies are used to ensure that integrity constraint violations do not take place in the database.
iv. Classes are mapped to tables in the following way:
   a. Each class is mapped to a table with the class name as the table name.
   b. Tables corresponding to independent classes are given a column named *classnameID* that serves as the primary key for the table. It is of type INTEGER.
   c. Association classes or Union of classes are given multiple columns, each referring via a Foreign Key to the classes that together make them. The primary key of the table is the composition of all the columns that form the class.
   d. Subclasses refer to the Superclass using a Foreign Key constraint.
   e. Assertions/Triggers are written to ensure Integrity Constraint Violations do not take place: Specifically, a BEFORE INSERT trigger is written to INSERT the data *first into the* super-class *and then* into the subclass. Similarly with association classes, data is first inserted into all of the tables that it refers to, and then into the table corresponding to the association class.
v. Datatype Properties are mapped to tables in the following way:
   a. Each datatype property is given a separate table that is named with the same name as the data type property. Two columns are created: *ClassID* which references the class to which the property belongs, and *Value* which contains the value.
   b. If the property is functional, it implies each individual of the class can have only a single value. This is enforced by making the *ClassID* column of the table the primary key.
   c. If the property is multivalued, it implies that each individual can take multiple values for the same property. This is enforced by making the primary key of the table a combination of both *ClassID* and *Value* columns.
   d. If the property is a sub-property of another property, a BEFORE INSERT assertion (or Trigger) is used in order to first insert the information into the super-property, and then into the sub-property.
vi. Object Properties are mapped to tables in the following way:
   a. Each object property is mapped to a table with the same name as the object property. The table has two columns, named *Domain* and *Range* that reference the primary key columns of the tables corresponding to the classes linked by the property.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

b. If the property is *functional* and *inverse functional* it means the function is a bijection. This is enforced by making the *Domain* column the primary key, and imposing a UNIQUE constraint on the *Range* column.
c. If the property is *functional*, but its inverse is not functional, it implies an individual in the *Domain* can have only a single *Range,* whereas the inverse, i.e. *Range* can have multiple corresponding domains. This is enforced by making the *Domain* column the primary key.
d. If the property is *inverse functional*, but is not *functional*, it implies an individual in the *Range* can have only a single inverse *Domain,* whereas the *Domain* can have multiple corresponding ranges. This is enforced by making the *Range* column the primary key.
e. If the property is neither *functional* nor *inverse functional* then there can be multiple allowed values for both *Domain* and *Range.* In this case, the primary key of the table is combination of both *Domain* and *Range* columns.
f. Please note that after mapping a property, the inverse of the property is implicitly mapped. It is not required to create separate tables for both the property as well as its inverse.
g. If the property is a sub-property of another property, a BEFORE INSERT assertion (or Trigger) is used in order to first insert the information into the super-property, and then into the sub-property.

IV. Mapping Semantic Constraints: OwlToRDBMS

The OWL constructs [17] that are most commonly used in modeling real-world systems include the properties listed in Table I. Additionally, we have also implemented some other important properties, which are also enumerated in Table I.

TABLE I
OWL CONSTRAINTS COMMONLY USED IN MODELLING REAL-WORLD SYSTEMS

allValuesFrom
someValuesFrom
hasValue
Cardinality
TransitiveProperty
SymmetricProperty
FunctionalProperty
InverseOf
InverseFunctionalProperty
AssymetricProperty
ReflexiveProperty
Irreflexive Property

Restrictions and other related OWL constructs are mapped to RDBMS using triggers/assertions. As listed above for Object Properties, there are four distinct cases to be dealt with. Table II provides an abstract definition of how the properties look under each of the four scenarios. In order to enforce the constraints, a two-pass system is required. For constraints such as minCardinality, someValuesFrom and Cardinality we cannot rely on INSERT/DELETE assertions. This is because these state that the number of individuals must be of a minimum number, i.e. after reaching that minimum number, the user must not be allowed to further delete individuals. This, however, cannot be checked during the initial building of the database, as Triggers do not have look ahead capability to foresee later insertion of records that will satisfy the constraints.

In-order to deal with this shortcoming, the application program itself enforces these constraints before creation of the DDL commands, and the triggers are written as BEFORE DELETE triggers to prevent later violation of the constraints at later times. It is important at this point to examine some of the difficulties in implementing triggers. In particular, most database management systems (if not all) do not allow triggers to insert data into the same table on which it is defined. As a work-around, three techniques are proposed:

i. A separate stored procedure is written that encapsulates both the insert procedure as well as the trigger, and the final commit is done after both statements are executed. This, however is not a viable option in our scenario as we require the data to be accessed dynamically via a pre-written stored procedure. Methods (b) and (c), however, are viable.
ii. A secondary READ_ONLY table is created and maintained. Every time data is written into the WRITE (object property) table, the trigger writes twice into the READ_ONLY table, once with the original assertion, and once with the inferred assertions. This implies, however, the user must execute WRITE statements into one table, and READ statements separately from another table.
iii. The final alternative is the creation of a separate *blackhole* table. A *blackhole* table is an optimized structure that accepts data and throws it away, and does not store it. A "BEFORE INSERT" trigger is written for the *blackhole* table so as to insert the incoming data back into the original table. After it is inserted into the original table, the *blackhole* table then discards the incoming data, therefore providing a memory efficient solution.

For constraints of the type someValuesFrom, hasValue and allValuesFrom, two modes of operation are possible:

i. The application program enumerates and hard-codes the possible/acceptable values into the trigger itself, which then checks it BEFORE INSERTION and raises an ERROR if the inserted values do not match the allowed values.
ii. In the case where the number of possible values is large, the above technique reduces readability and modifiability of the trigger. The alternative, therefore, is for the application program to create separate enumeration tables containing the allowed values, and the trigger referencing these tables using the IS IN SQL query. This allows for changing the values dynamically as well.

Finally, we describe the logic behind the implementation of the trigger procedures corresponding to every semantic constraint:

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

i. ***allValuesFrom***- The semantics of this restriction imply that for an entity to be valid, it must belong to a specified Class. It is analogous to the universal (for-all) quantifier of Predicate logic - for each instance of the class that is being described, every value for P must fulfill the constraint. This is implemented by writing a BEFORE INSERTION assertion that verifies whether the value being inserted into the table also has an entry in the table corresponding to the class of which it must be a part. This is done using an IS IN SQL query.

ii. ***someValuesFrom***- The semantics of this restriction imply that for a group of entities to be valid, at least one entity within the group must belong to a specified Class. It is analogous to the existential quantifier of Predicate logic - for each instance of the class that is being defined, there exists at least one value for P that fulfills the constraint. The implementation if this it two-fold. Firstly, while we are initially building the DDL commands from the ontology, the application program must check if this constraint is fulfilled, and is done with the help of OWL reasoners. Once the initial constraint is fulfilled, the database can be created. Now, however, we need to ensure that the constraint is not violated further ahead in time. The only case in which the
constraint can be violated, is on a DELETE SQL command, and therefore an assertion is written to check the validity of the constraint after a DELETE operation is carried out. If the constraint is found to be violated, an error is thrown, thereby preventing the DELETE operation from taking place.

iii. ***Cardinality***- There are three cardinality restrictions that we must map into assertions, namely minimum cardinality, maximum cardinality and cardinality (fixed cardinality.)
Like someValuesFrom, the minimum cardinality semantic restriction has a two-fold implementation. It is checked by the application program prior to the creation of the database, and an assertion is written to ensure that the constraint is not violated on deletion of records from the table. Similarly, assertions are written for maximum cardinality and fixed cardinality, an example of which is given below.

```
CREATE ASSERTION max_cardinality
        CHECK (SELECT COUNT(*) FROM
  table_2 > max) NOT DEFERRABLE;
```

iv. ***Transitive Property***- In order to implement the transitive property on entities, we need to use a separate table to make the inferred insertions, as most RDBMS do not allow triggers to insert records into the table on which they are written. Therefore, AFTER INSERT of the tuple (x,y) into the intended table T1, a search is carried out in T1, to retrieve all tuples of the form (d,x) where d is a generic record. For each record that matches this search, (d,y) is inserted into a dummy/black-hole table T2. Similarly, a search is carried out for records of the form (y,d) and (x,d) is inserted into T2 corresponding to the search results. T2 has a BEFORE INSERT trigger, which inserts all values into T1, thereby achieving the intended effect of inserting the inferred values of the transitive property into table T1.

v. ***Symmetric Property***- In order to implement the symmetric property on entities, we need to use a separate table to make the inferred insertions, as most RDBMS do not allow triggers to insert records into the table on which they are written. Therefore, AFTER INSERT of the tuple (x,y) into the intended table T1, (y,x) is inserted into a dummy/black-hole table T2. T2 has a BEFORE INSERT trigger, which inserts all values into T1, thereby achieving the intended effect of inserting the inferred values of the symmetric property into table T1.

vi. ***Reflexive Property***- In order to implement the reflexive property on entities, we need to use a separate table to make the inferred insertions, as most RDBMS do not allow triggers to insert records into the table on which they are written. Therefore, AFTER INSERT of the tuple (x,y) into the intended table T1, (y,y) and (x,x) are inserted into a dummy/black-hole table T2. T2 has a BEFORE INSERT trigger, which inserts all values into T1, thereby achieving the intended effect of inserting the inferred values of the reflexive property into table T1.

vii. ***Irreflexive Property***- In order the implement the irreflexive property, we need to invalidate all entries of the form (x,x), which is done using an assertion.

The final step in the process is mapping of the actual data. Each individual in the ontology is mapped to a row in the database. However, since the restrictions are NOT DEFERRABLE, the order in which they are inserted is also important. In order to ensure the generated SQL statements are in the appropriate order, the following technique is used. The structure of the tables created, specifically the foreign key references, is used as meta-data to process the individuals and create a directed acyclic graph. Topological sort is carried out on the graph, and if any dependencies are found to be unresolved, the application program asks the user to resolve them. This is the minor way in which our mapping process is semi-automatic as it relies on the user to resolve any discrepancies that may arise from the mapping process.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE II
MAPPING OF PROPERTY CONSTRAINTS TO RDBMS
(PK- Primary Key, FK- Foreign Key, ref- References)

Property between classes (mapped to table) A and class (mapped to table) B	*(Single Valued AND Optional) and has (Single Valued Inverse) implies B has FK ref PK(A)*	*(Single Valued) AND (Inverse is not Single Valued) implies A has FK ref PK(B)*	*(Multi Valued) AND (Inverse is Single Valued) implies B has FK ref PK(A)*	*(Multi Valued) AND (Inverse is Multi Valued) implies new table with FK ref A & B*
Transitive Property *implies* same domain and range	Not allowed as domain is single valued, hence (a,b) &(a,c) invalid	Not allowed as domain is single valued, hence (a,b) &(a,c) invalid	Not allowed as range is single valued, hence (b,c)&(a,c) invalid	On *insert* of (x,y): 1. Search for (d,x) and insert (d,y) and 2. Search for (y,d) and insert (x,d). PK = (id,op)
Symmetric Property *implies* same domain and range	On insert of (a,b): Insert on update (b,a) – *Check for integrity constraint*	Not allowed: if (a,b) & (c,b) exist *implies* (b,a) & (b,c) - invalid	Not allowed: if (a,b) & (c,a) exist *implies* (b,a) & (a,c) - invalid as (c,a) and (b,a)	On insert of (a,b): Insert (b,a)
Asymmetric property *implies* Same domain and range	On insert of (a,b): Check if (b,a) exists in table, invalidate if true	On insert of (a,b): Check if (b,a) exists in table, invalidate if true	On insert of (a,b): Check if (b,a) exists in table, invalidate if true	On insert of (a,b): Check if (b,a) exists in table, invalidate if true
Reflexive property	Reflexive property not possible as (a,b) implies (a,a)	Reflexive property not possible as (a,b) implies (a,a)	Reflexive property not possible as (a,b) implies (b,b)	On insert of (a,b): Insert (a,a) and Insert (b,b)
Irreflexive property	On insert of (a,a): Invalid	On insert of (a,a): Invalid	On insert of (a,a): Invalid	On insert of (a,a): Invalid
Cardinality Restriction	Cardinality of *property* : 0 or 1 Cardinality of *inverse* : 1	Cardinality of *property* : 1 Cardinality of *inverse*: find min/max/fixed cardinality, select count(*) < or > or =	Cardinality of *property* : find min/max/fixed cardinality, select count(*) < or > or = Cardinality of *inverse* : 1	Cardinality of *property* and of *inverse* : find min/max/fixed cardinality, select count(*) < or > or =

V. Implementation Details and Future Work

Our system is currently implemented using Apache Jena API which currently does not support IRI representation. A separate module was written in OWL API to retrieve the information that Jena cannot. Alternatively, an editing environment like Protégé can also be used to convert the OWL file into a suitable format that Jena can read.

Our future work includes porting the entire codebase to OWL API. Additionally, if the semantic constraints in the original ontology change, the database system needs to be regenerated. Our ongoing work involves developing dynamic mapping rules to write an incremental upgrade algorithm that would modify and run consistency checks only on parts that have changed to improve efficiency and reduce the overheads of checking semantic constraints in the database. It may be noted here that some prior work exists in dynamic mapping, particularly the work done by Yaun An et al. [18].

VI. Conclusion

In this paper, we have presented a complete and detailed technique to map a wide range of semantic constraints from an ontology to a conventional RDMS. Using our technique, we are able to use a combination of structural constraints, trigger procedures and assertions to bring the expressibility and flexibility of the Semantic Web into the scalability of a conventional RDBMS. The representation of real-world semantic constraints comes naturally using the constructs of OWL ontologies, and we are able to encode these semantics using semi-automatically generated trigger procedures. We have provided a viable mapping algorithm, such that post mapping, we are able to generate a self-contained RDBMS that is fully capable of behaving according to the semantic constraints of the domain.

References

[1] Michael Hammer and Dennis McLeod, "Database description with SDM: A semantic Database model" in ACM Transactions on Database Systems, Vol 6. No.3
[2] OWL 2 Web Ontology Language Document Overview at http://www.w3.org/TR/owl2-overview/
[3] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax at http://www.w3.org/TR/owl2-syntax/
[4] Farheen Siddiqui and M. Afshar Alam, "Web Ontology Language Design and Related Tools: A Survey" in Journal of Emerging Technologies in Web Intelligence, Vol. 3, No. 1
[5] TopBraid Composer, Maestro edition at http://www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition/
[6] Irina Astrova, Nahum Korda, and Ahto Kalja, "Rule-Based Transformation of SQL Relational Databases to OWL Ontologies"
[7] Guntars Bumans, "Mapping between Relational Databases and OwL Ontologies: an example" in Scientific Papers, University Of Latvia, 2010. Vol. 756
[8] Nasser Alalwan, Hussein Zedan, François Siewe, "Generating OWL Ontology for Database Integration" in Third International Conference on Advances in Semantic Processing
[9] Guohua Shen, Zhiqiu Huang, Xiaodong Zhu, Xiaofei Zhao, "Research on the Rules of Mapping from Relational Model to OWL"
[10] Petros Papapanagiotou, Polyxeni Katsiouli, Vassileios Tsetsos, Christos Anagnostopoulos and Stathes Hadjiefthymiades, "RONTO: Relational To Ontology Schema Matching" in AIS SIGSEMIS BULLETIN 3 (3&4) 2006
[11] Nikolaos Konstantinou, Dimitrios-Emmanuel Spanos and Nikolas Mitrou, "Ontology And Database Mapping: A Survey Of Current Implementations And Future Directions" in Journal of Web Engineering, Vol. 7, No.1
[12] Shalom Tsurt and Carlo Zanrolo, "An implementation of GEM, supporting a semantic model on a relational backend"
[13] Ernestas Vyšniauskas, Lina Nemuraite, Rimantas Butleris and Bronius Paradauskas, "Reversible Lossless Transformation From Owl 2 Ontologies Into Relational Databases" in Information Technology And Control, Vol.40, No.4

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[14] Irina Astrova, Nahum Korda, and Ahto Kalja, "Storing OWL Ontologies in SQL Relational Databases" in International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:1, No.5
[15] Angelina Tzacheva, Tyrone Toland, Peyton Poole, and Daniel Barnes, "Ontology Database System and Triggers" in Advances in Intelligent Data Analysis XII, Lecture Notes in Computer Science Volume 8207, 2013, pp 416-426
[16] Y. Liu and P. McBrien, "SQOWL2: Transactional Type Inference for OWL 2 DL in an RDBMS", ;in Proc. Description Logics, 2013, pp.779-790.
[17] Hai Zhuge, Yunpeng Xing and Peng Shi, "Resource Space Model, OWL and Database: Mapping and Integration" in ACM Transactions on Internet Technology, Vol.8, No.4
[18] Yuan An, Xiaohua Hu, and Il-Yeol Song, "Maintaining Mappings between Conceptual Models and Relational Schemas" in Journal of Database Management, 21(3), 36-68

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)