Towards a Formal Modelling of Data Warehouse Systems Design

Isaac Nkongolo Mbala and John Andrew van der Poll

***Abstract*—** Some research works have proposed a hybrid-driven approach that combines the data-driven approach and the requirements-driven approach to address specifically the design of static aspects around data warehouse systems. The star- and snowflake structures are two prominent approaches for specifying data marts of a data warehouse schema, resulting from a UML design. Both these approaches allow for ambiguity owing to their inherent use of semi-formal notations. Using a case-study approach of a data mart, we investigate in this paper the feasibility of formally specifying a data warehouse star schema in Z. We show how possible ambiguities that could lead to inconsistencies are clarified in the formal specification. The specification formalism is also enhanced through the consideration of aspects around user experience.

Keywords— Case study, Data mart, Data warehouse, Formal methods, Star model, UML class diagram, Z notations

I. Introduction

The use of data warehousing implies the development of a data warehouse built of data marts (DMs) or operational databases with business intelligence (BI) embedded in the resultant structure. Similar to a view in an operational database a data mart may correspondingly be looked upon as a subset of a data warehouse, i.e. it could be one or more of the underlying operational databases [1]. The design of a data warehouse is rather different from that of transactional systems since it is based on the decision-making support process of a company [2] – [5] and as confirmed by [6], the design process is arguably the most significant operation in the successful construction of a data warehouse system.

Some authors have suggested UML as a standard approach for the design of data warehouse systems in order to represent the multidimensional model, made up of entities such as facts, dimensions and sub-dimensions or hierarchies [7] – [9]. UML has been used at the analysis and design phases of system development and it has been useful for the design of data warehouse systems using class diagrams. More specific to data warehousing the star schema and the snowflake schema (and the galaxy structure as a combination of these two) have been used to model the corresponding static aspects of a data warehouse. Like UML, the star- and snowflake specifications are susceptible to ambiguities (cf. natural language) owing to their inherent semi-formal (diagrammatic, graphical, etc.) notations [10].

In an attempt to address some of the ambiguities of semi-formal notations, the use of formal methods (FMs) has been suggested for specifying software systems. Although formal methods do not ensure absolute system correctness [10], they assist in enhancing confidence in the correctness of the system by providing formal notations using discrete mathematics based on set theory and formal logic [11] [12]. These formalisms allow for strong, unambiguous designs and consistent models [13], and may be used for analyzing, specifying and checking the behavior and properties of a system that is viewed as a collection of mathematical objects [14]. FMs can assist to discover, and thereby reduce errors that may not be readily evident in semi-formal specifications [10]. Therefore, by modelling a data warehouse star- or snowflake schema formally, possible concerns of ambiguities, inconsistencies and shortcomings may be addressed to avoid time-consuming proofreading afterwards and costly reworks after system implementation. Owing to the popular use of star- and snowflake structures for data warehouse schema design, the authors experimented with both these structures, and opted for the use of the simpler star structure to specify a data warehouse schema. This observation agrees with the work by [15], in which the star schema is suggested as the better approach to the snowflake schema for data warehouse design. We return to this aspect in Section III.A in this paper.

Numerous formal-method notations have been developed to address challenges in software specification and design and some of the prominent ones have been ASM, VDM, CSP, CCS, Z, and B, derived from Z [16]. Accordingly, methodologies for transforming informal (natural language) system requirements to a formal specification have been defined, amongst others by [17], in which a translation from natural language to UML structures to the Z specification language is proposed. Consequently, in this paper we adapt the [17] methodology to formally specify a data warehouse schema around a data mart, using a star notation. The formal notation used is Z [18], based first-order logic and a strongly-typed fragment of Zermelo–Fraenkel set theory [19].

Following the above introduction, the remainder of the paper is organized as follows: A brief literature review on related work in formalizing UML-based structures and Z-related specifications is given in Section II, followed by the methodology that identifies the steps of transforming a data warehouse star schema to a Z specification in Section III. In Section IV, the application of a Z specification on a case study

Manuscript received October 16, 2020. The University of South Africa (Unisa) Funding supported this work. Isaac Nkongolo Mbala is an MSc Candidate in the School of Computing, Unisa, Florida, South Africa.

John Andrew van der Poll is a Research Professor at the Graduate School of Business Leadership (SBL), Unisa, Midrand, South Africa.

through a data mart is presented. Conclusions and future work in this area are presented in Section V.

II. Literature Review

Related work on UML as a diagrammatic modelling notation to define semi-formal specifications to create abstract models of specific systems is given in [20]. The combination of UML with Z has been discussed in the literature by numerous researchers, amongst others [21] [11] [22] [10] [23].

[11] report on the specification of an inventory system by combining UML and Z to investigate possible inconsistencies, improve the reliability of the system, and downscale shortcomings in the subsequent system development. A systematic process helping to transform and verifying UML sequence diagrams into Z by developing a method to assist with capturing hidden semantics of the said sequence diagram is addressed in [22]. In [10] an enhanced framework has been proposed for the verification and validation of static aspects of safety critical systems for the analysis of UML class diagrams and the relationships among them, using Z. In [24] an approach has been suggested to facilitate the consistency between class- and sequence diagrams in a multi-view modelling context.

Following a comprehensive literature survey, we determined that some authors have investigated on the one hand, the use of only formal methods for data warehouse systems [16], while on the other hand, others have considered the use of UML as a standard approach for the design of data warehouse systems [25] [7] [8] and recommended the use of a methodology that may facilitate the formalization of the class diagram for data-warehouse design at the conceptual level. As indicated above, however, in this paper we suggest a star-schema-formal-methods methodology to facilitate the design of data marts in data warehouse systems, aimed at eliciting possible ambiguities and inconsistencies.

III. Methodology

Research work on the combination of techniques or approaches has always been an interesting and major area of research owing to the introduction of new technologies and development of automated software tools [22]. Following [26]'s Research Onion, the research philosophy in this paper is both interpretive and positivist – the qualitative literature has been interpreted during a survey, and the Z schemas are positivist in nature; the approach to theory development is inductive since a formal specification is to be developed; the research choice may be classified as simple mixed (qualitative star diagram and quantitative Z specifications); the strategy followed is that of a case study (refer Section IV); the time horizon is cross sectional – a specific period (snapshot) in time; and the data collection technique is that of a literature review. The overall meta process for developing a Z specification from a data warehouse star schema is given in the following section.

A. Data Warehouse Star Schema

According to [27], there exist several models or approaches applied for the design of data warehouse systems based on multidimensional models. Among these models, three (3) are most often used for the design of a data warehouse. These are the star schema, snowflake schema and galaxy or fact constellation schema (a combination of the star and snowflake) but most research concur that the star schema and snowflake schema are the two most prominent approaches for data warehouse design, owing to their influence, and/or the advantages they offer to designers. A comparison of the two approaches (star schema and snowflake schema) has been made based on some parameters and a choice has been made on the star schema as the preferred one [15]. However, since the star schema also employs semi-formal notations through graphical structures to describe requirements it is susceptible to ambiguities. Consequently we define a simple methodology to provide formal notations to star schema specifications in order to address possible ambiguities, leading to inconsistencies.

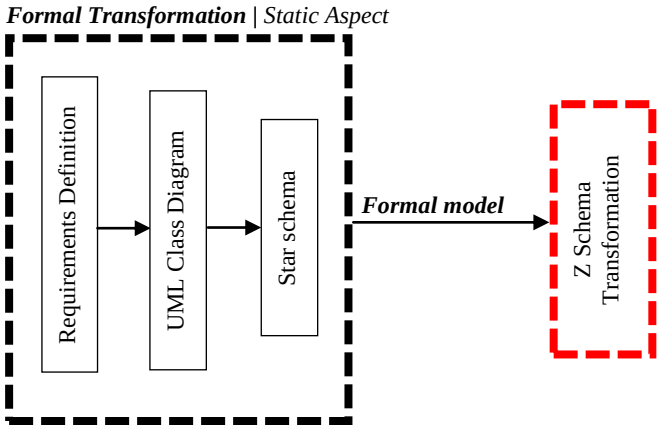

Fig. 1: Conceptual methodological steps

Fig 1 specifies the methodological steps for specifying a data warehouse schema. Traditionally system development starts out with a customer requesting the development of a bespoke system through a requirements definition which is usually in the form of a set of natural language requirements. These requirements are subsequently transformed into (usually) a semi-structured specification in the form of a set of UML (or similar notation) diagrams. The static part would then be translated into a snowflake- or star schema for the data marts of the warehouse. For this paper the star schema will be used. The star schema constructs are next translated into a formal (Z) specification, aimed at eliciting any ambiguities and inconsistencies.

B. Formal Z Specification

Z is a formal specification language based on a strongly typed fragment of Zermelo-Fraenkel set theory [19] and first-order logic. Owing to its set-theoretic roots, it embeds numerous discrete mathematical structures [11] [12]. It is arguably one of the most successful and widely used formal specification languages to describe and model computing systems. It furthermore has a formal (denotational) semantics [11]. In the opinion of the authors it is, compared to other formal notations, a simple and elegant specification formalism. Consequently, in this paper, Z is used for the purpose of formally specifying the static structures of a star schema denoting data warehouse marts.

The basic construct in a Z specification document is a schema, its generic format illustrated in Fig 2 [18]:

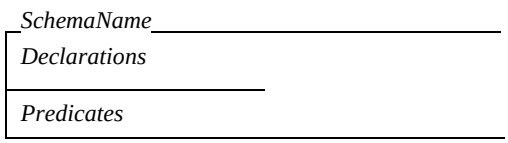

Fig. 2: Z Schema [18]

A Z schema generally consist of three parts: the name of the schema indicated at the top and two parts, namely, the declaration part containing the state variables (called components in Z terminology) and a predicate part that contains a set of predicates which constrains the variables and components mentioned in the declaration part [28].

Next, we introduce the case study for which a star schema and corresponding formal specification will be constructed.

IV. Case Study

A. Requirements Definition

Consider a company using a data warehouse system that offers car rental to customers from its different agencies at various locations. Information on the different agencies, customers, cars at the agencies, and the date at which transactions (reservations, etc.) take place should be maintained. Amongst other things, the decision makers are interested in the monetary value of a renting.

Suppose the data warehouse designer has to specify a data mart for the above system utilizing a star schema as per the following requirements:

1. *Define all entities/objects that would be involved in the design of the above car rental support system.*
2. *Describe all attributes for each entity/object and the relationships between entities/objects.*

B. A Car Rental System

Fig 3 shows the data warehouse star diagram that may be generated from the requirements of the case study in Section IV.1. The diagram consists of five (5) major star classes, namely, Agency, Car, Customer, Date and Renting, all in line with the requirements. Class Renting is further defined as a composite class having shared-aggregation relationships (terminology borrowed from a UML class diagram) with the other corresponding classes [25]. It should be noted that a data warehouse star schema utilizes constructs familiar to a UML class diagram in terms of objects/classes, relationships among the star classes, and constraints on the relationships.

The diagram in Fig 3 portrays a selection of the notation available in a data warehouse star construct, e.g. the use of aggregation (hollow diamond). Being the definition of a data warehouse and not an underlying operational data base, it typically would not utilize simple relationships like association (binary or otherwise). This aspect is addressed further in this paper.

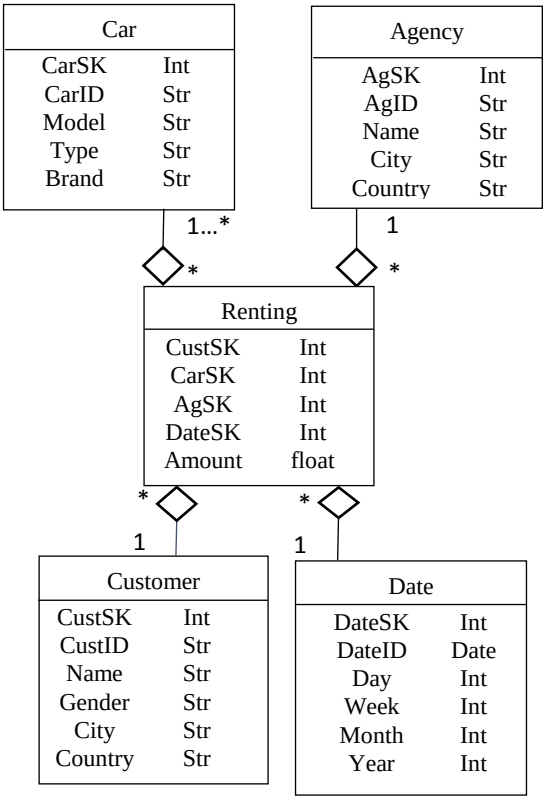

Fig. 3: UML Class Diagram for Car Rental System [25]

Further discussion of the star diagram and its inherent differences with a standard UML class diagram that would (e.g.) be constructed for one of the underlying operational databases (data mart) are in order.

1. An additional class, namely, Renting to maintain the various operations of the agency has been added to the four (4) classes. These are *agencies*, *customers*, *cars* at the agencies, and the *dates* of transactions (reservations, etc.) alluded to in the requirements definition above. In star-based terminology, a class like Renting in Fig 3 is known as a fact table while the other four classes are known as dimension tables. A fact is defined as a composite class having shared-aggregation relationships with the corresponding dimension classes. It is customary for fact classes to participate with corresponding dimension classes in aggregation relationships as indicated in the diagram [25].

2. The star schema defines two special attributes, loosely indicated by "SK" and "ID" in every dimension class. In traditional (relational) database terminology, the "ID" attribute would serve as the primary key for the relation and this requirement is upheld in the four (4) dimension classes. The "SK" attribute in a data warehousing context is a system-generated identifier, usually defined as an integer by the system described in Fig 3. Recall that a data warehouse includes a number of data marts or operational databases and it is possible that, for example,

a specific customer with a unique primary key occurs multiple times in various rentings on the same day. Consequently, the "SK" attribute keeps track of these customer occurrences, even those who have been deleted, since a data warehouse also keeps histories for business intelligence considerations [15]. From a data warehouse perspective, the "ID" primary key in the underlying database then becomes just a normal attribute in the dimension classes.

3. Still with Fig 3, the Renting class has an *aggregation* (hollow diamond) relationship with each of the four (4) dimension classes. In the underlying database(s), such relationships would mostly be *compositions* (filled diamonds), e.g. there would be a composition between Customer and Renting, indicating that if a customer demises, the renting record for such customer would be removed from the database. But since the data warehouse also records historical information, the relationship between Customer and Renting is an aggregation (hollow diamond) in Fig 3.

Next, we present a Z specification of the star diagram in Fig 3.

C. Formally Specifying the Star Schema in Z

In the process of translating a star diagram into a Z specification, the classes in the diagram essentially become Z schemas with additional restrictions as indicated in the generic version in Fig 2. For the sake of the user experience (UX) for the user of the specification, it is customary to use the same class names for schema names with some change in the letter face or font. Similarly, the attribute names are used in the corresponding schema. In line with the abstract characteristics of a specification, the specifier has the freedom to define the attributes types in a schema as deemed appropriate. The specification below follows the established strategy (ES) for constructing a Z specification [30] as well as the structure suggested by [21] for combining Z and UML.

Following the established strategy (ES) for constructing a Z specification, the first step is to define the basic types to be used in the specification. Initially we define five (5) basic types for the *Car* class.

These are:

[*CARSK, CARID, MODEL, TYPE, BRAND*]

Car

id! : *CARSK*
carid : *CARID*
model : *MODEL*
type : *TYPE*
brand : *BRAND*
rentings : ℙ *Renting* /* Set of rentings for a car to provide for historical information */

$\forall\ i, j : rentings \bullet\ i.\text{id!} = j.\text{id!} \Leftrightarrow i = j$

Fig. 4: Z Schema representing the Car Class

The attributes in the Car class in the star schema are indicated in Fig 4. As discussed in Section IV.B, unique identifiers are generated by the system to distinguish multiple, historical occurrences of an object. In Z output is indicated by a "!" decoration added to the variable name. An additional component, namely, *rentings* is defined as a set of renting instances for the particular car. This is done to allow historic information to be maintained in the data warehouse. The predicate in the schema specifies that renting identifiers generated by system are unique (generating a PO – proof obligation of course for a specification of such process).

Some information not readily evident in the Car class in the star diagram in Fig 3, is now explicit in schema *Car* in Fig 4 above: It is not evident that the denotation of attribute *CARSK* of an object of type Car in Fig 3 is system generated. But since Z explicitly allows for the decoration of variables (a system-generated output in this case) it is evident that id! in Fig 4 is system-generated, and not assigned by the user. This has implications for a correct algorithm underlying the generation of identifiers.

Standard Z has no notation for documentation (comments) inside a schema, yet to improve on the user experience of a schema we suggest adding documentation as indicated in the last schema predicate above. Likewise, while it is not customary in Z to provide a figure caption for a schema, we have done so, again to improve on the UX for the reader.

The Agency class in the star schema necessitates the introduction of further basic types, viz:

[*AGSK, AGID, NAME, CITY, COUNTRY*]

The Z schema for the Agency class is specified below:

Agency

id! : *AGSK*
agid : *AGID*
name : *NAME*
city : *CITY*
country : *COUNTRY*
rentings : ℙ *Renting*

$\forall\ i, j : rentings \bullet\ i.\text{id!} = j.\text{id!} \Leftrightarrow i = j$

Fig. 5: Z Schema representing the Agency Class

The system generates a unique agency id! and history for agency rentings is maintained.

New basic types for schema *Date* are:

[*DATESK, DATEID, DAY, WEEK, MONTH, YEAR*]

The schema for *Date* follows next:

Date

id! : *DATESK*
dateid : *DATEID*
day : *DAY*
week : *WEEK*
month : *MONTH*
year : *YEAR*
rentings : ℙ *Renting*

$\forall\ i, j : rentings \bullet\ i.\text{id!} = j.\text{id!} \Leftrightarrow i = j$

Fig. 6: Z Schema representing the Date Class

The basic types for schema *Customer* are given below, followed by the schema for Customer.

[*CUSTSK, CUSTID, NAME, GENDER, CITY, COUNTRY*]

Date

id! : *CUSTSK*
custid : *CUSTID*
name : *NAME*
gender : *GENDER*
city : *CITY*
country : *COUNTRY*
rentings : ℙ *Renting*

$\forall$ *i, j* : *rentings* • *i*. id! = *j*. id! $\Leftrightarrow$ *i* = *j*

Fig. 7: Z Schema representing the Customer Class

The Z schemas above show the formalization of the four (4) dimension classes and the single fact table in Fig 3. In each case, a unique identifier is generated by the system to identify multiple occurrences of objects maintained for historical purposes.

Next, we define the fact table Renting in Fig 3 in Z.

Renting

id! : *RentingID*
carsk : *CARSK*
custsk : *CUSTSK*
agsk : *AGSK*
datesk : *DATESK*
amount : *AMOUNT* /* Refer English prose discussion below schema */
cars : ℙ *Car* /* A customer may simultaneously reserve more than one (1) car */
customer : *Customer*
agency : *Agency* /* Assuming we are considering one (1) agency only */
date : *Date*

---

cars ≥ 1 /* *At least one* (1) *car is involved in a renting* */
$\forall$ *i, j* : *rentings* • *i*. id! = *j*. id! $\Leftrightarrow$ *i* = *j*

Fig. 8: Z Schema representing the Renting Class

Schema *Renting* represents the formalization of the aggregate object structure which in our case is the the renting class in Fig 3. As before for the dimension classes, the fact table formalized as *Renting* embeds a unique identifier generated by the system. The fact that at least one car has to participate in a renting transation is explicitly specified in the Z schema by # *cars* ≥ 1, a requirement that could be viewed as merely implicit in the star diagram in Fig 3 (the 1…* requirement between Car and Renting). Further explanation of the schema content is as indicted in the documentation.

Next, we turn to the formalization representing the constraint between the renting class and the car class as specified in the schema *RentalViewStruct* below. The predicate constraints portray the view of historical information maintained by the warehouse, but also instances that were created in the system but not yet destroyed. The formalization of the view for the Renting aggregation in Fig 3 consists of schema definitions for *Renting*, *Agency*, *Car*, *Customer* and *Date* previously specified.

Schema *RentalViewStruct* is specified as:

RentalViewStruct

rentings : ℙ *Renting*
cars : ℙ *Car*
customers : ℙ *Customer*
dates : ℙ *Date*
agencies : ℙ *Agency*

---

cars ≠ ∅ /* 1st predicate */

/* 2nd predicate */
$\forall$ r_1 , r_2 : *renting* | (r_1.*customer* = r_2.*customer*) ∨
(r_1.*agency* = r_2.*agency*) ∨
(r_1.*date* = r_2.*date*) ∨
(r_1.*cars* = r_2.*cars*) ⇒ $r_1 = r_2$

/* 3rd predicate */
$\forall$ *r* : *rentings* • (*r.agency* ∈ *agencies* ∧
r.customer ∈ *customers* ∧
r.date ∈ *dates* ∧
r.cars ⊆ *cars*)

/* 4th set of predicates */
$\forall$ *c* : *cars* • (∃ *r* : *rentings* • *c* ∈ *r.cars*)
$\forall$ *a* : *agencies* • (∃ *r* : *rentings* • *r.agency* = *a*)
$\forall$ *cs* : *customers* • (∃ *r* : *rentings* • *r.customer* = *cs*)
$\forall$ *d* : *dates* • (∃ *r* : *rentings* • *r.date* = *d*)

/* Uniqueness predicates */
$\forall$ a_1 , a_2 : *agencies* • a_1.id = a_2.id ⇔ $a_1 = a_2$
$\forall$ cs_1 , cs_2 : *customers* • cs_1.id = cs_2.id ⇔ $cs_1 = cs_2$
$\forall$ d_1 , d_2 : *dates* • d_1.id = d_2.id ⇔ $d_1 = d_2$
$\forall$ r_1 , r_2 : *rentings* • r_1.id = r_2.id ⇔ $r_1 = r_2$
$\forall$ c_1 , c_2 : *cars* • c_1.id = c_2.id ⇔ $c_1 = c_2$

Fig. 9: Z Schema representing the Class Diagram

As captured in Fig 3, the *Renting* class forms aggregations with all four (4) dimension classes. This requirement is captured in the declarations section in schema *RentalViewStruct.* The 1st predicate states that at least one *Car* participates in the system. The 2nd predicate specifies that all valid *rentings* link to corresponding *agency*; *customer*; *car*; and *date* objects (i.e. any two items of *rentings* are the same Renting instance should they have at least one part in common).

As per the 3rd predicate all valid rentings (*r* : *rentings*) have these as elements of the defined sets (*agencies*, *customers*, *dates* and *cars*) in the system., i.e. all parts of the *Renting* instance come from the sets of existing instances. The 4th predicate set specifies that *Car* instances may be shared among instances of *Renting* owing to the many-to-many relationships existing between the classes and their multiplicities. Included in this predicate set are constraints that all created instances of the part-of classes should be parts of created aggregate instances. The uniqueness predicates state that identifiers previously generated by the system for each dimension-class object and each object in the fact class are unique.

It is evident that schema provides additional, and also more explicit information than may be inferred from the star schema

in Fig 3. This reduces ambiguity that may lead to inconsistencies further on in the design of such system.

V. Conclusions and Future Work

In this paper, we presented an overview of a data warehousing system. Two prominent structures, namely, the snowflake- and the star schemas were briefly addressed. Owing to its simpler structure, the star schema was selected to specify a data-warehouse case study. A brief methodology for moving from an informal specification to UML structures to a star schema, and ultimately a Z specification was defined. The major purpose of formal methods (as captured in a formal specification) in assisting designers to specify and design more reliable systems was unpacked. Chief among these were eliciting possible ambiguities and inconsistencies in non-formal specifications, especially during the requirements gathering and early specification phases.

Next a Z specification of the static structures as captured in the star schema around a data mart was presented. Some amendments to the formal specification to facilitate UX for users of the specification was put forward. Aspects around the aggregation of four (4) dimension classes and one fact class formed part of the formalism. Implicit (or absent) information in the star diagram was elicited in the Z schemas, thereby revealing hidden information and eliminating ambiguity.

Future work in this area may be pursued along a number of avenues: Dynamic aspects around a star schema should be investigated and these should be specified for the data warehouse snowflake schema as well. It would prove insightful to see whether the relative simplicity of a star schema translates into a correspondingly simpler Z specification than for a snowflake structure. The specification of a view (schema *RentalViewStruct*) leads to a complex schema and tool support to investigate simpler schemas that are logically equivalent to the said schema should be considered.

References

[1] I. N. Mbala and J. A. van der Poll, "Towards Specification Formalisms for Data Warehousing Requirements Elicitation Techniques". *The 3rd International Conference on Computing Technology and Information Management (ICCTIM 2017)*, (1), 45–58, 2017.

[2] S. Mathur, G. Sharma and A. K. Soni, "Requirement Elicitation Techniques for Data Warehouse Review Paper". *International Journal of Emerging Technology and Advanced Engineering*, *2*(11), 456–459, 2012. Retrieved from https://www.researchgate.net/profile/Girish_Sharma4/publication/259827907_IJETAE_1212_84/links/02e7e52e0dc392211f000000.pdf

[3] N. H. Z. Abai, J. H. Yahaya and A. Deraman, "User Requirement Analysis in Data Warehouse Design: A Review". *Procedia Technology*, *11*, 801–806, 2013. https://doi.org/10.1016/j.protcy.2013.12.261

[4] M. El Mohajir and I. Jellouli, "Towards a Framework Incorporating Functional and Non Functional Requirements for Data Warehouse Conceptual Design". *IADIS International Journal on Computer Science and Information Systems*, *9*(1), 43–54, 2014. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.640.5590&rep=rep1&type=pdf

[5] A. Nasiri, E. Zimányi and R. Wrembel, "*Requirements Engineering for Data Warehouses"*. 49–64, 2015. Retrieved from http://code.ulb.ac.be/dbfiles/NasZimWre2015incollection.pdf

[6] R. Jindal and T. Shweta, Comparative Study of Data Warehouse Design Approaches : A Survey. *International Journal of Database Management Systems*, *4*(1), 33–45, 2012. https://doi.org/10.5121/ijdms.2012.4104

[7] S. Mann, "*Object Oriented Multidimensional Model for a Data Warehouse with Operators"*. *3*(4), 35–40, 2010.

[8] W. Tebourski, W. Ben, A. Karâa and H. B. Ghezala, "*Semi-Automatic Data Warehouse Design methodologies : A Survey"*. *10*(5), 48–54, 2013.

[9] M. Thenmozhi and K. Vivekanandan, "Data Warehouse Schema Evolution and Adaptation Framework Using Ontology". *International Journal on Computer Science and Engineering (IJCSE)*, *6*(07), 232–246, 2014.

[10] M. Singh, A. K. Sharma and R. Saxena, "An UML + Z Framework for Validating and Verifying the Static Aspect of Safety Critical System". *International Conference on Computational Modeling and Security*, 352–361, 2016. https://doi.org/10.1016/j.procs.2016.05.243

[11] S. H. Bakri, H. Harun, A. Alzoubi and R. Ibrahim, "the Formal Specification for the Inventory System Using Z Language". *The 4th International Conference on Cloud Computing and Informatics*, (064), 419–425, 2013.

[12] M. Rodano and K. Giammarco "A Formal Method for Evaluation of a Modeled System Architecture". *Procedia - Procedia Computer Science*, *20*, 210–215, 2013. https://doi.org/10.1016/j.procs.2013.09.263

[13] F. Valles-barajas, *Using Lightweight Formal Methods to Model Class and Object Diagrams*, 2012. https://doi.org/10.2298/CSIS110210045V

[14] A. Adesina-Ojo, "Towards the Formalisation of Object-Oriented Methodologies". University of South Africa, 2011. https://doi.org/10.1145/2072221.2072252

[15] K. I. Mohammed, "Data Warehouse Design and Implementation Based on Star Schema vs . Snowflake Schema". *International Journal of Academic Research in Business and Social Sciences*, *9*(14), 25–38, 2019. https://doi.org/10.6007/IJARBSS/v9-i14/6502

[16] J. Q. Zhao, "*Formal Design of Data Warehouse and OLAP Systems", 2007*. Retrieved from http://mro.massey.ac.nz/handle/10179/718

[17] M. Lall, "*A Methodology for the Formalization of Non-Functional Requirements of Web Services"*, PhD Thesis, School of Computing, University of South Africa (Unisa), South Africa, 2013.

[18] J. M. Spivey, "*The Z Notation : A Reference Manual"*. Prentice-Hall International, 1992.

[19] H. B. Enderton, "Elements of Set Theory", Academic Press Inc, 1977. https://doi.org/10.1016/S0049-237X(08)71114-5

[20] N. Ibrahim and R. Ibrahim, "Semantic Rules of UML Specification". *Proceedings of MUCEET2009*, 37–40, 2009.

[21] M. Shroff and R. B. France, "Towards a Formalization of UML Class Structures in Z". *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97)*, 646–651, 1997. https://doi.org/10.1109/CMPSAC.1997.625087

[22] N. A. Zafar, Formal Specification and Verification of Few Combined Fragments of UML Sequence Diagram. *Springer - Arab J Sci Eng*, *41*, 2975–2986, *2016*. https://doi.org/10.1007/s13369-015-1999-9

[23] S. A. Han and H. Jamshed, "Analysis of Formal Methods for Specification of E-Commerce Applications. 35(1), 19–28, 2016.

[24] K. EL Miloudi, Y. EL Amrani and A. Ettouhami, "Using Z Formal Specification for Ensuring Consistency in Multi-View Modeling". *Journal of Theoretical and Applied Information Technology*, *57*(3), 407–411, 2013.

[25] S. Luján-mora, "*Data Warehouse Design with UML PhD Thesis"*. University of Alicante, 2005.

[26] M. Saunders, P. Lewis, A. Thornhill and A. Bristow, "*Research Methods for Business Students"*, Pearson, 6th edition, 2019, Pearson. ISBN: 9781292208787

[27] B. P. Basaran, A Comparison of Data Warehouse Design Models. *Analyzer*, 2005.

[28] E. S. Grant, "Towards an Approach to Formally Define Requirements for a Health & Status Monitoring for Safety-Critical Software Systems". *Lecture Notes on Software Engineering*, *4*(3), 2016. https://doi.org/10.18178/lnse.2016.4.3.244

[29] K. El Miloudi, "A Multiview Formal Model of Use Case Diagrams Using Z Notation : Towards Improving Functional Requirements Quality". *Journal of Engineering*, 1–10, 2018. https://doi.org/https://doi.org/10.1155/2018/6854920

[30] B. Potter, J. Sinclair and D. Till, "An Introduction to Formal Specification and Z", (2nd edition), Prentice Hall, 1996.

Isaac N. Mbala is an MSc Candidate in Computer Science with School of Computing, University of South Africa (UNISA), South Africa. He obtained his BSc Hons (Computer Science) from the University of Kinshasa (UNIKIN). He has worked for a telecomm company as Management Information System (MIS) Specialist from 2013 to 2015 in the DRC. He was in charge of Business Intelligence, Development and Reporting. He also worked for JB Capital Partners company as IT Specialist – Software Engineer from 2019 to 2020. He was in charge of Coding, Testing and deployment of the software applications.

His current focus area is to determine the extent to which the use of formal methods for data warehouse systems may alleviate failures within the design process. He is looking forward to do his PhD in Computer science.

Prof John A. van der Poll obtained a BSc from the University of Stellenbosch, South Africa in 1980 and a Hons BSc (Computer Science) in 1982, also from Stellenbosch. He obtained an MSc (Computer Science) from Unisa (University of South Africa) in 1987 and a PhD in Computer Science from Unisa in 2001. He was employed in the Unisa School of Computing from 1988 to 2013. He became a full Professor in Computing in 2007 and moved to Unisa's Graduate School of Business Leadership (SBL) in Midrand in June 2013.

His research interests are in the construction of highly dependable Business software, specifically the formal specification and subsequent reasoning of the properties of Business Intelligence applications, Data warehousing, the IoT, IT Governance, and aspects of the 4th Industrial Revolution. He is a Research Professor at the SBL, an NRF rated researcher, category C2 and supervised numerous Master's and Doctoral students to completion.