Using Model-Driven Engineering for Decision Support Systems Modelling, Implementation and Powering

Apollinaire Bamana Batoure *apollinaire.batoure@univ-ndere.cm*
University Institute of Technology
The University of Ngaoundere
Ngaoundere, Cameroon

Kolyang *kolyang@univ-maroua.cm*
Higher Teachers' Training College
The University of Maroua
Maroua, Cameroon

Abstract

Following the principle of *everything is object*, software development engineering has moved towards the principle of *everything is model*, through Model Driven Engineering (MDE). Its implementation is based on models and their successive transformations, which allow starting from the requirements specification to the code's implementation. This engineering is used in the development of information systems, including Decision-Support Systems (DSS). Here we use MDE to propose an DSS development approach, using the Multidimensional Canonical Partitioning (MCP) design approach and a design pattern. We also use model's transformation in order to obtain not only implementation codes, but also data warehouse feeds.

Keywords: Model Driven Engineering, Multidimensional Canonical Partitioning, Multidimensional Design Pattern, Model Transformation.

1. INTRODUCTION

Within the framework of information systems realization, the Object Management Group (OMG) proposed the development by models [1] called Model-Driven Engineering (MDE). This engineering allows the assisted and controlled passage between models, with four abstraction levels. Transformations are possible between and within models [2]. At each of these levels or models, specific actions to be performed are established. The exclusive use of models and transformations is what makes MDE different from the classical information systems developing approach.

In this paper, we use model-driven engineering to coordinate the overall process of modelling (conceptual, logical and physical), implementing and powering a decision system. For this purpose, we use previous results, namely:

- the multidimensional partitioning approach [3, 4] which allows to design a multidimensional data schema in six steps;
- a multidimensional and spatio-temporal design pattern, which will guide the generation of the data schema [5, 21].

After this introduction, we will bring up system development through models and their use in decision-making systems. We will then describe the actions carried out within the different phases of the proposed architecture. It will be presented at last, just before the conclusion.

2. DEVELOPMENT THROUGH MODELS

Following the object approach and its *everything is an object* principle, software engineering has moved towards the *everything is a model* principle, through *Model Driven Engineering (*MDE) [6, 7]. It is a discipline that places models at the centre of software engineering processes. The basic idea is the separation of the functional specifications of a system from the details of its implementation on a given platform [8]. Its implementation is entirely based on models and their transformations [9, 10].

In mathematics, a model is a translation of reality in order to apply tools, techniques and theories to it. In computer science, its objective is to structure data, processing and information flows between entities [11]. A model is an abstract description of an entity in the real world, using a given formalism. It will be used first to model the application and then, by successive transformations, to generate the implementation code [8].

In MDE, a model describes real-world instances. The model is itself described by a language called a meta-model. The language on languages is called meta-meta-model. It is self-described. We thus deduce an architecture with four abstraction levels, in addition to the description of the instances of the real world. This relationship is illustrated by *Figure 1*, adapted from the *MDA Guide, version 2* [12]. More concretely, a model represents, through a system, a field reality. The model is described by a legend, which is a meta-model. The legend itself follows a number of rules that constitute the meta-meta-model or language on language.

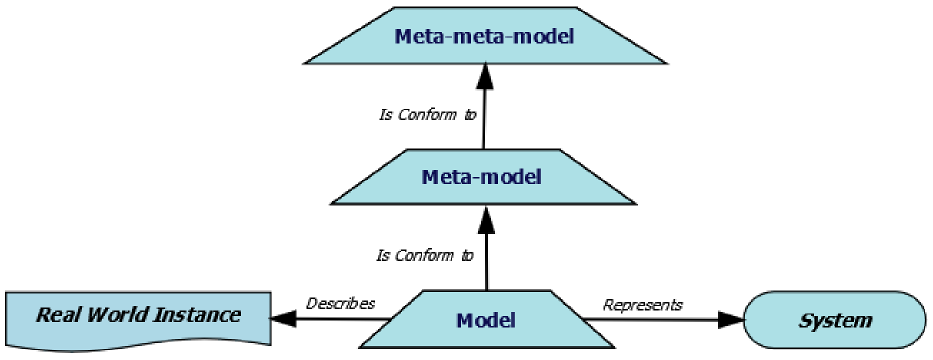

FIGURE 1: Levels of Model Abstraction.

The advantages of a model are that it is *abstract*, *understandable*, *accurate, predictive*, *timesaving*, *reusable*, *extensible, flexible* and *documented* [6, 13]. To this can be added *productivity*, *portability*, *interoperability, easy maintenance* and *documentation* [14]. The model-based development approach makes the quotation *model once and generate anywhere* concrete [10, 15]. To do so, it proposes an application development based on the elaboration of a *platform-independent model* and its transformation into a *model dependent* and a *specific platform*, on which the system is implemented and deployed [2]. For its implementation, model-based development is made up of several standards [2, 7]. These standards aim at perpetuating the different models. We will use in this work, the QVT (*Query View Transformations*) standard [6, 11, 12].

In the development process, everything is considered as a model, both the data schemas and the source code. Thus, the code is no longer a central element, but the result of the model transformation. These models or levels of this engineering are [6, 7]:

- *Computation Independent Model* (CIM) or business requirements model;
- *Platform Independent Model* (PIM) or analysis and design model;
- *Platform Specific Model* (PSM) or detailed design and code model;
- *Platform Description Model* (PDM) or description of the system runtime environment.

The implementation of all these models and the associated transformations makes it possible to define a **Y** development cycle [16]. The passage between the different models is achieved by transformations. Model transformation is the process that converts one model into another model of the same system. Three types of transformations are identified [2, 7, 9]. They are:

- vertical (*CIM → PIM, PIM → PSM*)
- horizontal (*PIM → PIM, PSM → PSM*)
- or reverse (*code to PSM, PSM to PIM and PIM to CIM*).

Reverse transformations build models from existing applications and thus implement reengineering techniques.

3. MODELS IN THE DEVELOPMENT OF DECISION SUPPORT SYSTEMS

With regard to the modelling of *Decision Support Systems* (DSS), the first model-driven proposal was done in 2003 [16]. This work proposes an approach based on the different packages of the *Common Warehouse Meta-model* (CWM**)** and from a UML formalism. Most of the work that followed was proposed around 2010. Some approaches simply provide a set of steps to guide the designer in the development of the conceptual model. Others present in addition, the logical model and eventually the physical model [14].

Zepada et *al.* in [17] proposed a mixed modelling approach summarized in three phases. The first phase is devoted to the examination and reorganization of the Entity/Relationship (E/A) schemas of the legacy database, in order to determine the multidimensional elements. In the second phase, the needs of the company's users (or decision-makers) are taken into account. The last phase integrates the two points of view and generates the solution supporting the source of data that best reflects the needs of the decision-makers. Rules for transforming E/R meta-models into OLAP (On-Line Analytical Processing) meta-models are defined.

The work of *Essaidi* and *Osmani* ([18]) brings together *Unify Processing* (UP) engineering and *model engineering* (MDE) for the design and development of DSS. Model engineering is used for system design within an integrated and standard framework. While process engineering, through 2TUP (*2 Track Unified Process*), is used for the iterative development of the decision-making system.

Mazon and *Trujillo* ([19]) use model engineering throughout the data warehouse development process. To do so, they defined the UML (*Unified Modelling Language*) and CWM (*Common Warehouse Meta-model*) profiles needed to describe these models. The QVT language is used for transformations and source code generation.

Mazon and *Trujillo*, this time in [20], proposed a hybrid approach based on model engineering for the design of BI (*Business Intelligence*) systems. It involves five steps, which are:

- the definition of information needs at the CIM level;
- the derivation of an initial PIM from the CIM;
- taking into account existing data and systems to generate a hybrid PIM;
- the CWM is used to define the PSM tailored to the different database technologies;
- finally, the implementation of the various QVT between models, and generation of data warehouse codes.

Atigui et *al* ([13]) model the decision-making system by including the associated ETL (*Extraction-Transformation-Loading*) processes. They take into account the complete development cycle, i.e. needs analysis, conceptual, logical and physical modelling in addition to ETL processes. The proposed approach is said to be mixed, unified and automatic. Conceptual modelling is done using a UML profile for data warehouses. Logical modelling is done through a software choice of ROLAP (*Relational OLAP*), MOLAP (*Multidimensional OLAP*), HOLAP (*Hybrid OLAP*) or XML, depending on the case. The physical modelling is implemented by materialized views that facilitate calculation, storage, automatic update and refresh through the chosen DBMS. The QVT language is used for model's translation.

Hachaichi et *al.* in [15] proposed a model-driven approach based on the use of multidimensional patterns. The multidimensional pattern is used for the expression of requirements. Model engineering is used to coordinate the implementation cycle of the DSS.

The MDE development approach we propose in this work, takes into account a design approach for DSS [3, 4], a multidimensional design pattern [5] and transformations, according to the QVT language [23, 24]. The transformations are defined for the transition between the CIM to PIM and then to PSM. This approach has the advantage of modelling both the decision system and the feed processes. In the following section, we will describe the actions at the different phases of engineering, before presenting the proposed architecture.

4. ACTIONS AT THE REQUIREMENTS LEVEL

The *Computation Independent Model* (CIM), or requirements model, is the business model. It helps to represent exactly what the system should do, without specifying the details of how. Its purpose is to promote understanding of the problem and to establish a common vocabulary. The requirements expressed in this model help in the construction of the successor models (PIM and PSM).

Thus, this level aims at producing the multi-dimensional data schema, which will then be refined and implemented. It further describes *step 6 of* the MCP (*Multidimensional Canonical Partitioning*) approach [3]. It consists of generating the data schema from the multidimensional elements (dimensions, hierarchies, fact tables) obtained in *steps 4* and *5 of the MCP* approach. For this, we use the elements of the multidimensional *annotation* (AM), the *design pattern* and the *QVT language for* model transformations. From these elements, the *design pattern (*PM) and the *transformation language* (QVT), the ready-to-implement *data store* (MD) is generated. *Figure 2* illustrates the transformations to be performed in *step 6 of the* multidimensional canonical partitioning approach, in order to output the specific computational model or data store (MD).

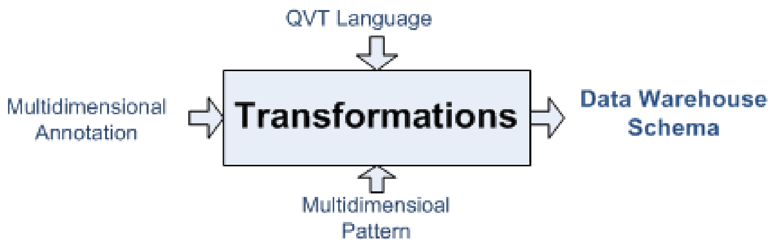

FIGURE 2: Actions at CIM Level.

Query View Transformation (QVT) is a declarative language standardized by the OMG. A QVT transformation between two candidate models is specified through a set of relationships. Each transformation is composed of the following elements [13, 17, 19, 23, 24]:

- *Domains*, which refers to a candidate model and a set of elements to be linked;
- *Domain Relationship*, which specifies the type of relationship between domains. It can be marked as *Checkonly* (C) or *Enforced* (E).

A *Checkonly* domain is used to check whether there is a valid match that satisfies the relationship. While an *Enforced* domain allows you to create an element in the model, if the match link is checked.

The *When* clause describes the preconditions that must be met in order to carry out the transformation. While the *Where* clause determines the post-conditions to be met by all model elements participating in the relationship.

A transformation contains two types of relationships: *top-level* and *non-top-level*. The execution of a transformation requires that all *top-level relations* are executed, whereas *non-top-level relations* must be executed when they are invoked, directly or transitively from the *Where* clause of another relation.

4.1 Description of The Transformation Rules

The initial model is the *multidimensional annotation* (**AM**), and the target model is the *data store* (**MD),** instantiated from a design pattern, defined in [5]. The elements of multidimensional annotation are:

- dimensions + dimension attributes + associations to facts or to hierarchies;
- hierarchies + attributes of hierarchies + association to dimension or other hierarchy;
- facts + measures.

A formal description of the elements of multidimensional annotation is given by *Atigui* et *al.* in [13].

We start with the transformations from dimension to dimension table, then from hierarchy to hierarchy table, and finally from facts to facts table. The transformations rules are defined as follows:

R1: Transformation of dimension into dimension table

- Each dimension in **AM** is transformed into a dimension table in **MD** + primary key (CP);
- Each dimension attribute in **AM** is transformed into a dimension attribute in **MD**;
- The spatial and temporal dimensions are taken into account in a particular way, following the archetype of the multidimensional pattern [5].

R2: Transformation of hierarchy into hierarchy table

- Each hierarchy in **AM** is transformed into a hierarchy table in **MD** + primary key;
- Each hierarchy attribute in **AM** is transformed into a hierarchy attribute in **MD**;
- A referential relationship is created with the other hierarchy tables and/or the dimension.

R3: Transformation of facts into table of facts

- Each fact in **AM** is transformed into a table of facts in **MD** + primary key;
- A referential relation is created between the fact table and the corresponding dimensions;
- Each fact measure in **AM** is transformed into a fact attribute in **MD**.

The transformations to be executed are thus *Main*, *DimensionToTable*, *HierarchyToTable* and *FactToTable*.

4.2 Carrying Out The Transformations

The meta-models used for transformations are **MetaModelAM** (initial model) for multidimensional annotation and **MetaModelMD** (target model) for the data warehouse. The QVT language programs, corresponding to the transformations, are shown in *Table 1.*

The programs presented in these tables are model transformations. The first one is the principal (*main*). It describes how to switch from **MetaModelAM** to **MetaModelMD**. In the *checkonly domain* block, the elements of the initial model are declared. In the *enforce domain* block, the elements of the target model are declared. The *where* clause contains the procedures defined as *top relation,* which will be executed as needed.

In the other procedures (*DimensionToTable, FactToTable and HierarchyToTable*), one declares the initial elements (*checkonly domain)* and the target elements (*enforce domain)*; the possible preconditions (*when) and* post-conditions (*where) to be* checked for the transformation to take place.

Transformation AM2MD (in am : MetalModelAM, out md : MetalModelMD) name : : string ;
top relation main {
checkonly domain {
am : : MetalModelAM ;
name = n ;
Dimension = amd : : MetalModelAM ;
Hierachy = amh : : MetalModelAM ;
Fact = amf : : MetalModelAM ;}
enforce domain {
md : : MetaModelMD ;
name = n ;
Dtable = td : : MetaModelMD ;
Htable = th : : MetaModelMD ;
Ftable = tf : : MetaModelMD ;}
where
{DimensionToTable (amd, td) ;
HierarchyToTable (amh, th) ;
FactToTable (amf, tf) ;}
}
	Top relation HierarchyToTable {
	dimtable = dt : : MetaModelMD ;
	hierTable = ht' : : MetaModelMD ;
	checkonly domain { hname = n ;
	hierarchy = amh : : MetaModelAM ;
	hierForeignKey = hfk : : MetaModelAM ;
	hierAtt = ha : : MetaModelAM ;
	parameter = p : : MetaModelAM ;}
	enforce domain { tname = n ;
	hiertable = ht : : MetaModelMD ;
	hiertablefk = htfk : : MetaModelMD ;
	hiertableatt = hta : : MetaModelMD ;
	rel = RelationShip (a, b) ;}
	when {
	RelationShip (dt, ht) OR RelationShip (ht', ht) ;}
	where {
	ParameterToPrimaryKey (p, htfk) ;
	HierAttToHierTableAtt (ha, hta) ;
	HierFKToHierTableFK (hfk, htfk) ;}
	}

Top relation DimensionToTable {
checkonly domain { dname = n ;
parameter = amp : : MetaModelAM ;
dimatt = da : : MetaModelAM ;
hierarchy = h : : MetaModelAM ;}
enforce domain {
tname = n ;
dimtable = dt : : MetaModelMD ;
primarykey = pk : : MetaModelMD ;
dimtableatt = dta : : MetaModelMD ;}
where {
ParameterToPrimaryKey (amp, pk) ;
DimAttToDimTableAtt (da, dta) ;}
}
	Top relation FactToTable {
	dimtable = dt : : MetaModelMD ;
	checkonly domain { fname = n ;
	fact = amf : : MetaModelAM ;
	measure = amm : : MetaModelAM ;}
	enforce domain { tname = n ;
	facttable = ft : : MetaModelMD ;
	foreignkey = fk : : MetaModelMD ;
	primarykey = pk : : MetaModelMD ;
	factatt = fa : : MetaModelMD ;}
	when {RelationShip (dt, ft) ;}
	where {MeasureToFactAtt (amm, fa) ;
	ParameterToForeignKey (dt.pk, fk) ;}
	}

TABLE 1: Transformation QVT Programs.

The transformations are also described schematically. Thus, in the diagrams of figures 3, 4, 5 and 6, the four transformations are described by their initial elements, those of target, then pre- and post-conditions.

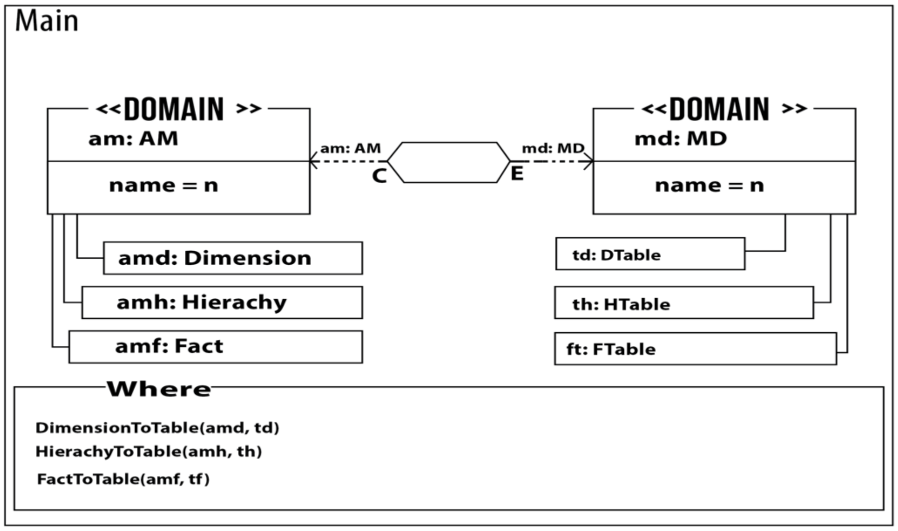

FIGURE 3: Description of The *Main* Transformation.

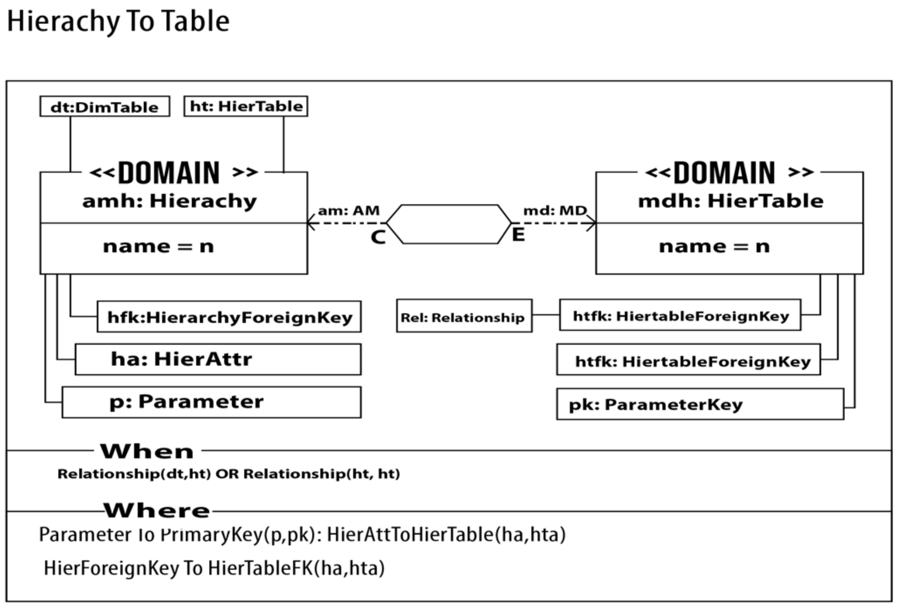

FIGURE 4: Description of The *HierarchyToTable* Transformation.

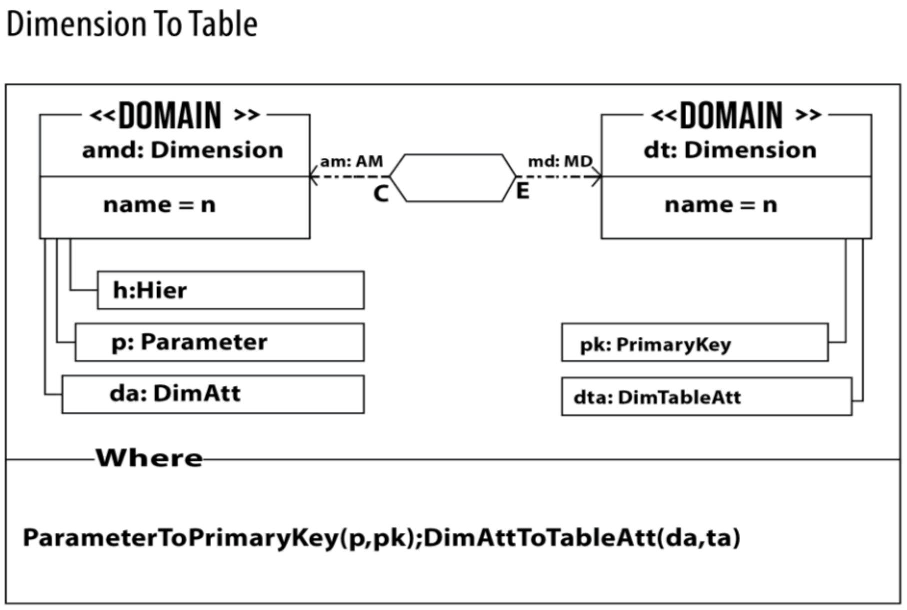

FIGURE 5: Description of The *DimensionToTable* Transformation.

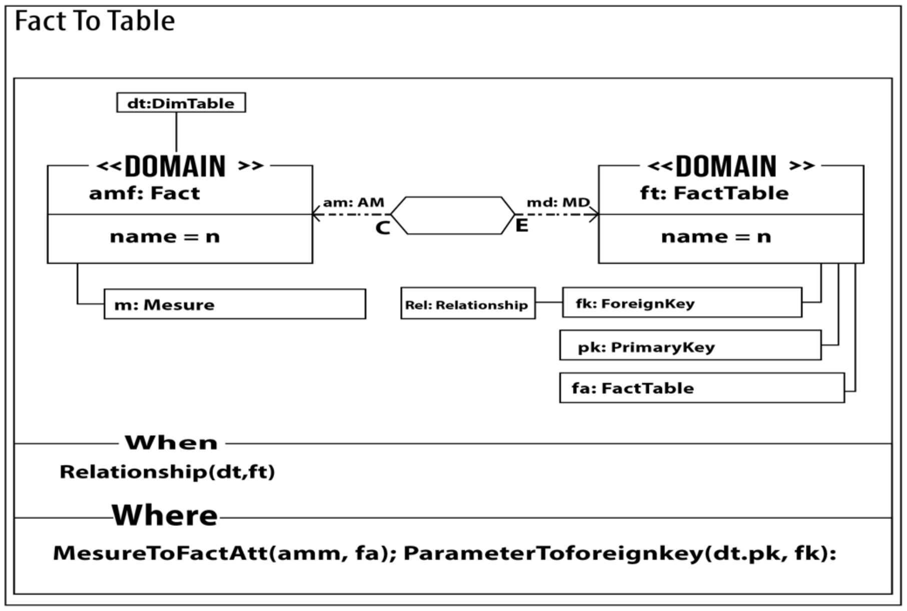

FIGURE 6: Description of The *FactToTable* Transformation.

These QVT codes have thus enabled us to carry out the necessary transformations at the CIM level. The resulting multidimensional data schema is the input element of the PIM level.

5. ACTIONS AT OTHER LEVELS OF ARCHITECTURE

The other levels are *Platform Independent Model* (PIM), *Platform Dependent Model* (PSM) and description of platform used, called *Platform Description Model* (PDM).

5.1 Platform-independent Actions

The *Platform Independent Model* (PIM), or analysis and design model, defines the structure and behaviour of the system. It describes the system without showing the details of its implementation on a given platform. It will be refined by the specificities of one or more particular architecture(s).

At the input of this level, we have a data schema that best solves the problem posed from the previous level. This schema is refined at the level of data types, their domains and constraints, the different triggers and the implementation techniques retained. For this, several types of meta-models or implementation models can be used. These implementations are *Relational OLAP*, *Multidimensional OLAP*, *Hybrid OLAP* or *Dynamic OLAP* models. Here we use the ROLAP meta-model, to which we will add the spatio-temporal aspects.

The transformations at this level are the transition from the conceptual multidimensional schema to the logical multidimensional schema. We use the cardinalities of the associations between entities and primary and/or referential integrities. The transformations rules are summarized below. They are inspired by the passage from a conceptual data model to a relational model [22]. These rules are:

- each entity becomes a table;
- each property of an entity becomes an attribute of the table;
- the entity identifier becomes the primary key of the table;
- the mesh links (1..n or 0..n) result in the creation of a new entity;
- hierarchical links (0..1 to 0..n or 1..n) result in key migration.

After the transformation into a logical data schema, the types, domains and constraints on each attribute are specified. This phase produces a data dictionary. At the output of the PIM level, we have a data schema ready to be implemented on a chosen platform.

5.2 Platform-dependent Actions

The *Platform Specific Model* (PSM), or detailed design and code model, is the projection of a PIM onto a given platform. The PSM is generated for each technology platform. It also includes code generation, optimization, compilation, packaging, initialization and system configuration. A PSM is thus generated from a PIM. To do this, it is based on the platform used, as well as its description. In this model, we take into consideration the ready-to-implement logical data schema from the previous level.

Transformation at this level is the transition from the logical model to the implementation and feed codes of the data warehouse. It is done according to the chosen platform. The data warehouse is implemented in a relational DBMS. A multidimensional server is used for the analyses and views in the form of hypercubes. Feeding processes are developed in a specific environment, using ETL tools. Thus, the entire deployment of the data warehouse and its feed processes can be done.

5.3 Action At The Platform Description Level (PDM)

The *Platform Description Model* (PDM), on the other hand, describes the environment on which the system will be implemented and run. It allows us to describe, through manuals, tutorials and others, the use of the selected platforms. In this case, we have a DBMS (Relational and Spatial), a multidimensional database server, an ETL environment, the query and programming languages necessary to implement, deploy and feed the data warehouse.

The following section presents the overall architecture for the development of the decision-support system, through the models we propose.

6. DEVELOPMENT ARCHITECTURE

The set of actions to be carried out, at the level of the different models, as presented above, allows us to deduce the diagram in *Figure 7*.

The architecture presented has four levels. These levels are those of model engineering. At each level, the actions carried out there are presented. These are the input elements, the transformations, the results obtained, or the implementation choices made.

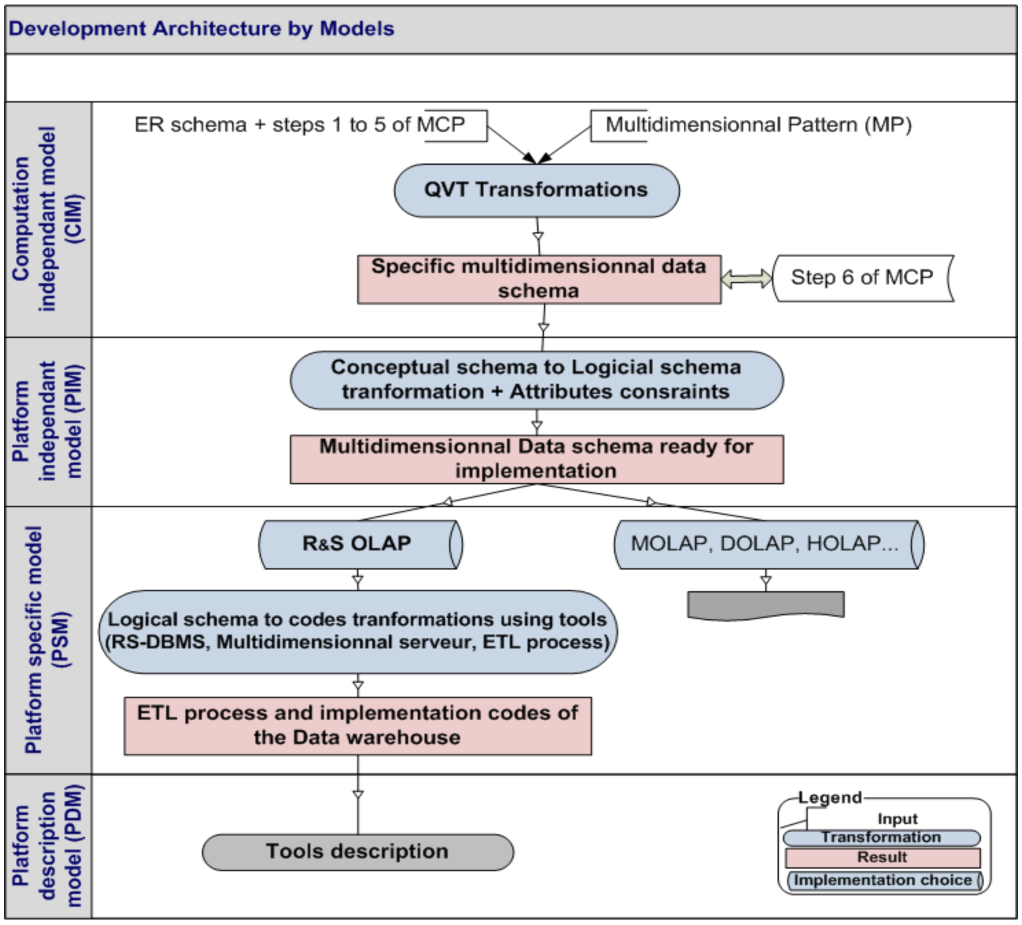
Development Architecture by Models
Computation independant model (CIM)
ER schema + steps 1 to 5 of MCP
Multidimensionnal Pattern (MP)
QVT Transformations
Specific multidimensionnal data schema
Step 6 of MCP
Platform independant model (PIM)
Conceptual schema to Logicial schema tranformation + Attributes consraints
Multidimensionnal Data schema ready for implementation
Platform specific model (PSM)
R&S OLAP
MOLAP, DOLAP, HOLAP...
Logical schema to codes tranformations using tools (RS-DBMS, Multidimensionnal serveur, ETL process)
ETL process and implementation codes of the Data warehouse
Platform description model (PDM)
Tools description
Legend
Input
Transformation
Result
Implementation choice

FIGURE 7: Model-Based Development Architecture.

The summary of this diagram is as follows:

- at the *CIM level*, the multidimensional elements from the first five steps of the MCP approach and the design pattern are taken as input. Using QVT languages, the multidimensional data schema is obtained. This is equivalent to step 6 of the MCP approach;
- at the *PIM level*, the conceptual data schema obtained previously is transformed into a logical data schema, ready for implementation. The implementation is done according to a technological choice of the decision domain. Among these possibilities, we choose the relational and spatial decision model;
- from the logic diagram and the technological model, transformations are carried out at the *PSM level*, in order to obtain the codes for both implementation and data supply. The tools needed here are a *relational and spatial DBMS*, a *multidimensional server* and the *powering processes*;
- The *PDM level* allows the description of the different tools that are used.

At the end of the application of these phases, we have an operational, fed, powered and ready for operating data warehouse.

The approach we proposed, comparing to other one such as [13, 15, 17, 18, 19, 20, 25, 26, 28, 29], is original because it takes into account:

- A new DSS modelling approach ([3, 4]);
- Spatio-temporal data ([5]);
- And uses model driven engineering (present works).

Furthermore, it takes into consideration, in the same approach, the modelling, development, implementation and powering of the DSS. Afterward, we can apply on the obtained DSS, data mining and knowledge extraction technics.

7. CONCLUSION

The present work ends a series of four papers on a decision-support system approach. It combines the multidimensional canonical partitioning approach, multidimensional pattern and model engineering, to propose an approach for the design, development, implementation and powering of decision-support systems. The developed decisional systems take into account both attribute and spatio-temporal data. This is why the approach can be applied to all domains, using spatio-temporal reference data. Next steps will lead us to apply the whole approach on spatio-temporal domain, such as the one of urban sector.

8. REFERENCES

[1] V.G. Díaz, J.M.C. Lovelle, B.C.P. García-Bustelo, O.M. Sanjuán. “Advances and Applications in Model-Driven Engineering”. In ASASEHPC, IGI Global, 2014.

[2] Y. Rhazali, Y. Hadi and A. Mouloudi. “Model Transformation with ATL into MDA from CIM to PIM Structured through MVC”. In Procedia Computer Science, Vol. 83, pp. 1096-1101, 2016.

[3] A.B. Batoure, Kolyang and M. Tchotsoua. “Using Multidimensional Canonical Partitioning (MCP) as a Supply-Driven Approach for Data Warehouses Design”. International Journal of Computer (IJC), 25(1), 2017, pp. 52–62.

[4] A.B. Batoure and Kolyang. “Designing Decision Support Systems Approaches using Entity/Relationship Data Schema: A Survey”. International Journal of Scientific & Engineering Research (IJSER), Volume 9, Issue 9, September 2018.

[5] A.B. Batoure and Kolyang. “Multidimensional and Spatio-temporal Design Pattern to generate Data Warehouses schema”. International Journal of Computer Applications Technology and Research, Volume 9–Issue 06, 2020, pp. 211-216.

[6] X. Blanc and O. Salvatori. MDA in action: Model-driven software engineering. Eyrolles, Paris, 2011.

[7] L.G. Cretu, and F. Dumitriu. Model-Driven Engineering of Information Systems: Principles, Techniques, and Practice. Apple Academic Press, Inc., 2015.

[8] S. Diaw, R. Lbath, and B. Coulette. “State of the art on software development based on model transformations”. TSI Special Issue - Model Driven Engineering, 29(2), 2010, pp. 4-5.

[9] H. Kadima. MDA: Model Driven Object Oriented Design. Dunud, Paris, 2005.

[10] L.F. Pires, S. Hammoudi and B. Selic. “Model-Driven Engineering and Software Development”. In 5th International Conference, MODELSWARD 2017 Porto, Portugal.

[11] B. Combemale. “Model-driven engineering: state of the art. Technical report”. Institut de Recherche en Informatique de Toulouse, Toulouse, 2009.

[12] J.M. Siegel. MDA Guide revision 2.0. Technical report, OMG, Boston, 2014.

[13] F. Atigui, F. Ravat, O. Teste and G. Zurfluh. “A model-driven approach to the design of multidimensional data warehouses”. In IRIT (GIS/ED), 2010.

[14] F. Atigui. “Model-driven approach for the implementation and reduction of data warehouses”. PhD thesis, University of Toulouse, 2013.

[15] Y. Hachaichi, H. Ben-Abdallah and J. Feki. “Towards an MDA approach to data warehouse development”. In Systèmes d'Information et Intelligence Economique, pages 158-170, 2008.

[16] J.D. Poole. “Model Driven Data Warehousing (MDDW)”. Technical report, OMG Announces Integrate, Burlingame, California, USA, 2010.

[17] L. Zepada, L. Celma and R. Zatarain. “Mixed Approach for Data Warahouse Conceptual Design with MDA”. In International Conference on Computational Science and Its Applications (ICCSA’08), pages 1204–1217, 2008.

[18] M. Essaidi and A. Osmani. “Data warehouse development using MDA and 2TUP”. In 18th International Conference on Software Engineering and Data Engineering 2009, SEDE 2009, (1) :138–143, 2009.

[19] J.-N. Mazón and J. Trujillo. “An MDA approach for the development of data warehouses”. Decision Support Systems, 45(1) :41–58, 2008.

[20] J.-N. Mazón and J. Trujillo. “A hybrid model driven development framework for the multidimensional modelling of data warehouses”. ACM SIGMOD Record, 38(2), 2009.

[21] A. B. Batoure and M. Tchotsoua. “Use of open source software for the management of spatially referenced data”. Africa Science, 13(3) :377 - 389, 2017.

[22] J. Akoka and I. Comyn-Wattiau. Conception des bases de données relationnelles, en pratique. Vuibert, Collection Informatique, Paris, 2001.

[23] OMG, “MOF 2.0 Query/View/Transformation (QVT)”, V1.0. OMG Document – formal/08-04-03, 2008.

[24] I. Arrassen, A. Meziane, R. Sbai and M. Erramdani. “QVT Transformation by Modelling – From UML Model to MD Model”. International Journal of Advanced Computer Science and Applications (IJACSA), 2 (5), 2011, pp. 7-14.

[25] L.A. Fernandes, B.H. Neto, V. Fagundes, G.Z. da Silva, J.M. De Souza, R.S. Monteiro. “Model-Driven Architecture Approach for Data Warehouse”. Sixth International Conference on Autonomic and Autonomous Systems, ICAS 2010, Cancun, Mexico.

[26] F. Atigui, F. Ravat, R. Tournier and G..A Zurfluh. “Unified Model Driven Methodology For Data Warehouses And ETL Design”. InProceedings of the 13th International Conference on Enterprise Information Systems (ICEIS-2011), pages 247-252.

[27] J.-N. Mazón, J. Trujillo, M.A. Serrano, M. Piattini. “Applying MDA to the development of data warehouses”. DOLAP 2005, ACM 8th International Workshop on Data Warehousing and OLAP, Bremen, Germany, November 4-5, 2005.

[28] O. Betari, S. Filali, A. Azzaoui, M.A. Boubnad. “Applying a Model Driven Architecture Approach: Transforming CIM to PIM Using UML”. iJOE - Vol. 14, No. 9, 2018, pp. 171-181.

[29] A. Azzaoui, O. Rabhi, A. Mani. “A Model Driven Architecture Approach to Generate Multidimensional Schemas of Data Warehouses”. iJOE - Vol. 15, No. 12, 2019, pp. 19-31.