

Visual Modelling of Data Warehousing Flows with UML Profiles^{*}

Jesús Pardillo¹, Matteo Golfarelli², Stefano Rizzi², and Juan Trujillo¹

¹ Lucentia Research Group, University of Alicante, Spain

`{jesuspv,jtrujillo}@dlsi.ua.es`

² DEIS, University of Bologna, Italy

`{matteo.golfarelli,stefano.rizzi}@unibo.it`

Abstract. Data warehousing involves complex processes that transform source data through several stages to deliver suitable information ready to be analysed. Though many techniques for visual modelling of data warehouses from the static point of view have been devised, only few attempts have been made to model the data flows involved in a data warehousing process. Besides, each attempt was mainly aimed at a specific application, such as ETL, OLAP, what-if analysis, data mining. Data flows are typically very complex in this domain; for this reason, we argue, designers would greatly benefit from a technique for uniformly modelling data warehousing flows for all applications. In this paper, we propose an integrated visual modelling technique for data cubes and data flows. This technique is based on UML profiling; its feasibility is evaluated by means of a prototype implementation.

Keywords: OLAP, UML, conceptual modelling, data warehouse.

1 Introduction

Data transformations are the main subject of visual modelling concerning data warehousing dynamics. A data warehouse integrates several data sources and delivers the processed data to many analytical tools to be used by decision makers. Therefore, these data transformations are everywhere: from data sources to the corporate data warehouse by means of the ETL processes, from the corporate repository to the departmental data marts, and finally from data marts to the analytical applications (such as OLAP, data mining, what-if analysis). Data warehousing commonly implies complex data flows, either because of the large number of steps data transformations may consist of, or of the different types of data they carry. These issues rise interesting challenges concerning design-oriented modelling of data warehousing flows. In particular, the thorough visualisation of these models has a deep impact on the current trends for data

^{*} Supported by the Spanish projects: ESPIA (TIN2007-67078), QUASIMODO (PAC08- 0157-0668), and DEMETER (GVPRE/2008/063). Jesús Pardillo is funded by MEC under FPU grant AP2006-00332. Special thanks to the anonymous reviewers for their helpful comments.

warehousing design, where the so-called model-driven technologies [1] promote diagrams as the main, tentatively unique, design artefacts managed by software engineers. Cognitive aspects, such as diagrams readability, are thus related to the productivity of the whole development process.

Nevertheless, the main research efforts made so far have concerned the static modelling of the data warehouse repository [2] and, even when data warehousing flows were considered, it was done within specific business intelligence applications (OLAP, data mining and so on). While these efforts were addressed at designing individual modelling frameworks, all of them characterised nothing but data transformations.

Our contribution in this paper is twofold: firstly, we identify and formally define *data warehousing flows* (f^w 's) as the founding concept for every visual modelling technique studied so far, which means that flows can be applied to model any data transformation, from OLAP to data mining (§2). Due to space constraints, this paper is focused on OLAP f^w 's that, as a matter of fact, are the backbone of data warehousing. In general, an OLAP session model can be useful in different contexts: (1) it can be a relevant part of more complex data warehousing flows (e.g., it could describe a set of transformations to be applied to multidimensional data in a what-if analysis application); (2) it can be used to design or document a semi-static report, where a limited number of OLAP operators can be applied depending on the data currently visualized; (3) it can represent auditing information showing system administrators the more frequent operators applied to cubes.

Secondly, we present an integrated visual modelling framework for f^w 's based on UML [3]. The proposal consists in the diagramming of data cubes, meant as results of multidimensional queries (§3), and data transformations (§4). These diagrams have been implemented in a prototype illustrated in §5. Remarkably, our solution covers the modelling gaps identified in the state-of-the-art as it is shown in §6. Finally, conclusions are drawn in §7.

2 The f^w Framework for Visual Modelling

A data warehousing flow f^w may be functionally characterised as $f : I \rightarrow O$ where f is the (probably complex) data transformation, I is the universe of data objects managed by f and O is the universe of the processed data objects. Moreover, f is defined as the composition of other functions, $f = f_1 \circ \dots \circ f_n$, that may recursively be defined as compositions.

f^w 's may be classified according to the actual data-object type they manipulate, *i.e.*, the domain I and codomain O . Data cubes, we argue, are the most important factor in this classification because they are the building blocks of data warehouses. Let C be the universe of all data cubes involved in an f^w , and X denote any unspecified sort of data objects; we can distinguish the disjoint categories shown in Table 1.

For instance, *mining flows* may be characterised as f^{-c} , because they transform data cubes into other data objects, namely data-mining patterns such as

Table 1. General taxonomy of data flows based on their data-object types

Name	Notation	Definition
OLAP flow	f^c	$f^c : C \rightarrow C$
ETL flow	f^{+c}	$f^{+c} : X \rightarrow C$
mining flow	f^{-c}	$f^{-c} : C \rightarrow X$
object flow	f^\emptyset	$f^\emptyset : X \rightarrow X$

association rules or clusters. Though the names for flow types were chosen according to the field where they mainly appear, in practice they are not bound to a single application domain, such as in the case of what-if analysis, where different types may be involved.

The canonical f^w may be decomposed (regarding Table 1) into: $f^w = f^\emptyset \circ f^{-c} \circ f^c \circ f^{+c} \circ f^\emptyset$, where \circ is any composition operator and the f^w 's from the right to the left respectively characterise: (1) transactional flows for populating data sources; (2) ETL flows for populating the data warehouse; (3) OLAP flows occurring during an OLAP session; (4) mining flows aimed at extracting patterns from the data warehouse; and (5) flows for manipulating and visualising patterns. Though there are in practice many ways of connecting f^w 's (*e.g.*, multiple branches are valid structures), it is evident that f^w 's involving data cubes either in input or in output are the actual backbone of the data warehousing process. Noticeably, all f^w 's that involve data cubes might also be characterised as atomic f^\emptyset 's at the finest detail level, because cubes can be decomposed into their elements (dimensions, measures, etc.). For this reason, f^{-c} 's and f^{+c} 's will be sometimes visually modelled as a composition of detailed f^\emptyset 's instead of being considered as atomic. Conversely, visually modelling internal details of f^c 's is out of the scope of this paper.

Visual modelling of f^w 's should comply with the following wish-list: (i) it should be based on some multidimensional diagrams that model data warehouse facts and dimensions, (ii) it should be easy to understand, (iii) its semantics should have formal foundation, and (iv) it should rely on a standard notation.

The need to manage f^w complexity suggests to create separate diagrams for data cubes. For this reason, our framework provides two kinds of diagrams, namely *data cube* and *f^w diagrams*. Data cube diagrams represent a multidimensional query formulated on the data warehouse. f^w diagrams represent how actions transform data cubes.

For such diagrams to be cognitively effective, their notations should achieve a *conceptual integration* of information from separate diagrams into a coherent user's mental model and a *perceptual integration* by means of perceptual cues (orienting, contextual and directional information) in order to support navigation between diagrams [4]. In f^w diagrams, data cubes are just rendered as *information scents* [5] (like the scent of food) that encourage readers to look for more detailed diagrams (where more succulent information could be found). Conversely, data cube diagrams render each data cube in detail. In addition, data cubes are visually modelled over the *multidimensional diagrams* [6]

of the underlying data warehouse facts, thus providing the required perceptual cues¹. The following two sections describe data cube and f^w diagrams, respectively.

3 Data Cube Diagrams

Our aim is to propose a visual modelling technique based on standard representations of f^w 's. A well-known extension technique such as UML profiling [3] seems then appropriate. Profiling enables to easily but formally extend (in terms of both semantics and notation) the UML language, the *de facto* standard for general-purpose modelling in software engineering. By means of profiling, data cubes can be smoothly hosted in a UML multidimensional diagram describing the data warehouse. To accomplish this goal we will use the UML profile for multidimensional modelling presented in [6], namely the **DataWarehouse** UML profile, which also provides a proper iconography that improves diagram readability.

Fig. 1 illustrates a sample multidimensional diagram (*host diagram*) to which the data cube diagram of Fig. 2 is allocated (*guest diagram*). The host diagram represents the database for analysing the **sales** facts (⌘ in Fig. 1) by **product**, **customer**, **location**, and **date** (⌘). Each dimension allows sales to be aggregated (⊙) at different *granularities*. For instance, sales may be aggregated by **month** and **year** (⌘). In addition, facts and dimensions can be respectively described by measures (**M**, *e.g.*, **quantity**) and descriptors (**D**, *e.g.*, **city name**).

On the other hand, the guest diagram represents a query that may be easily referred to the host diagram. For instance, Fig. 2 shows a cube of sales **quantity** grouped by **month**, store **city** and product **branch**; in particular, only the branches whose code is food.

3.1 A UML Profile for the Integrated Diagramming of Data Cubes

Our **DataCube** profile is based on the **DataWarehouse** profile by Luján-Mora *et al.* [6]. Both profiles are represented in Fig. 3 with the standard notation for UML profiling [3], *i.e.*, the profile diagrams where stereotypes of UML modelling elements are presented. On the one hand, the **DataWarehouse** profile contains a set of *stereotypes* (*e.g.*, **Fact** and **Dimension**). Each stereotype represents a single multidimensional concept by extending the specific UML *metaclass* considered as the most semantically close to that concept. Fig 3 also shows the proper iconography. On the other hand, the **DataCube** profile (see Fig. 3) introduces five stereotypes²:

¹ Indeed, visual models contain two kinds of data: statements about the reality that they model and metadata about how they are represented, such as canvas locations.

² We chose names for the **DataCube** profile stereotypes according to the terminology of the multidimensional expressions (MDX) from Microsoft, the most spread language for OLAP querying, to emphasise that data cubes are the result of queries.

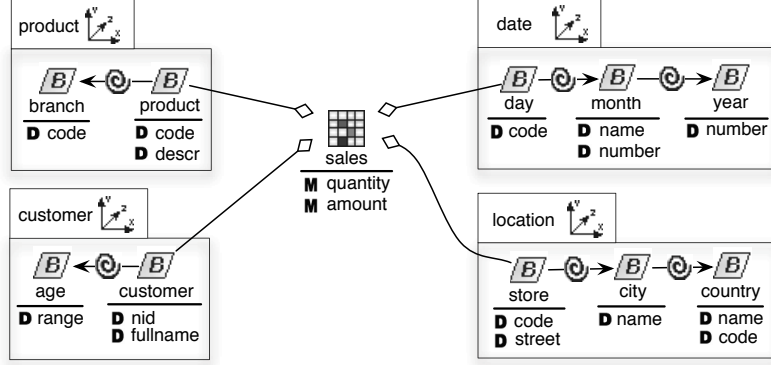


Fig. 1. Multidimensional diagram by using the DataWarehouse UML profile

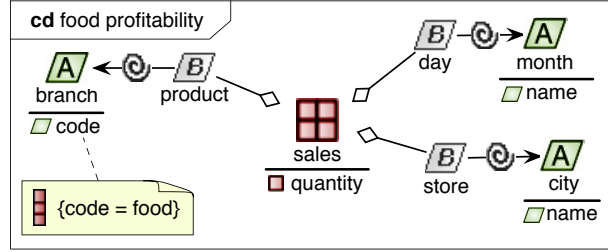


Fig. 2. Data-cube diagram by using the DataCube UML profile

Cell, summarising a set of Facts of a given cube.

Axis, showing each Base (component of the group-by clause) of a given cube.

CellMember & AxisMember, representing each returned Measure and Descriptor.

Slice, representing the predicate formulated on CellMembers and/or AxisMembers.

All these stereotypes specialise the **CubeElement** abstract stereotype. Every **CubeElement** has references to both the extended metaclass and the supporting entity of the multidimensional diagram (shown by a *use* dependency in Fig. 3). The **cube** attribute of the **CubeElement** stereotype is a tag definition referring to all the cubes that contain a cube element. Let F be a class stereotyped as **Fact**. In order to represent a cube c resulting from a query on fact F , you need to additionally stereotype F as **Cell** and annotate class F with tagged value c for the **cube** attribute of **Cell**.

All **CubeElements** (except **Slices**) have two abstraction levels:

Space Specification, where either **Cell** or **Axis** stereotypes are applied to **Facts** or **Bases**, respectively. Each retrieved cell or axis member are left to the designer as a *variation point* [3] whose options are: (i) all owned properties (measures or descriptors) are retrieved in that cell; or (ii) they still remain unspecified. Unless differently stated, the second option is assumed.

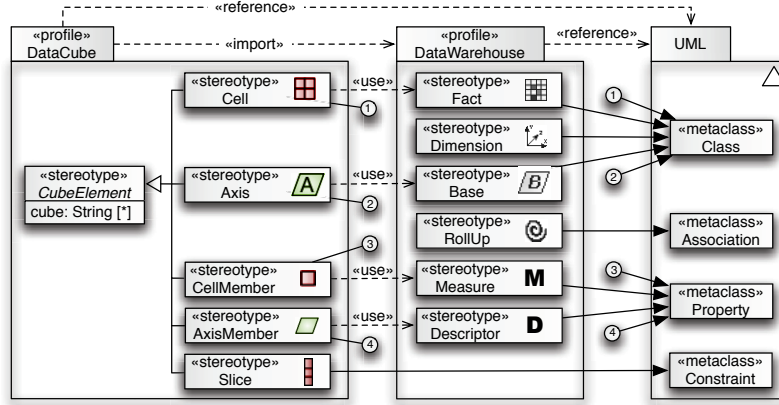


Fig. 3. DataCube and DataWarehouse UML profiles

Member Specification, where *CellMember* or *AxisMember* stereotypes must be applied together with the owner *Cell* or *Axis*, respectively. The underlying space specification is thus explicitly shown. The application of these stereotypes may be managed by diagramming tools as we shall discuss next.

The rules about how to apply and combine the proposed stereotypes are formally specified in OCL [7] (*i.e.*, a declarative language to query and specify constraints in UML models), and thus, they could be automatically managed by the corresponding checking engine. For instance, the OCL constraint to check the rule of *CellMember* specifications is

```
context CellMember inv 'Member Specification Rule':
self.base_Property.class.extension_Cell.cube->includesAll(self.cube)
```

3.2 Rendering Data Cube Diagrams

UML profiles allow to adapt the UML notation to include new iconography. In this way, the *DataCube* profile provides a new version of the *DataWarehouse* stereotypes whose aim is to emphasise the retrieved data. This marking is accomplished by swapping the *DataWarehouse* icons, rendered in grayscale, with new coloured versions. The aesthetics decision of colouring is justified by the cognitive studies about *preattentive processing* [8], stating that coloured diagram nodes are distinguished from grayscale ones before conscious attention occurs, thus showing *CubeElements* “at a glance”. In addition, the selected colours are *complementary* (red vs. green), so they can be distinguished very well from each other. Due to black-and-white printing, this iconography also uses shapes (resembling the underlying *DataWarehouse* elements) for codifying *CubeElements*. It is worth noting that there is not a *DataWarehouse* counterpart of the *Slice* stereotype, because particular predicates regarding specific data instances only concern data cube diagrams, not multidimensional diagrams.

Nevertheless, rendering **DataCubes** over **DataWarehouses** requires special customisations of diagramming tools. Since every **CubeElement** references all the cubes it belongs to, the **DataCube** iconography setting should be context-aware. The context is the current diagram itself: only the **CubeElements**, whose **cube** property includes the name given to the current diagram, are actually rendered with **DataCube** icons (though they may internally refer to other cubes). Of course, this assumes that both the **CubeElement::cubes** and diagram names refer to the same set of values, *i.e.*, the actual names of the data cubes being diagrammed.

3.3 Drawing Data Cubes

The workflow for drawing **DataCubes** consists of the following steps:

1. **Copying the DataWarehouse diagram**, representing the repository from which the data cube will be retrieved, into a new **DataCube-to-be** diagram.
2. **Renaming this DataCube diagram** in order to identify the desired cube.
3. **Specifying CubeElements** by following one of the two specifying conventions discussed above (namely, space or member) and by attaching the proper **Slices** to the corresponding **CubeElements**. This step is actually decomposed into (i) application of **CubeElement** stereotypes if they were not already applied for a previous data cube, and (ii) addition of the current diagram name as a **CubeElement::cube** tagged value. Since the second step could be sometimes cumbersome, it can be automatically managed in diagramming tools by implementing the corresponding controllers for context-aware marking.
4. **Hiding undesired DataWarehouse elements** (optional) for enhancing the final diagram readability. This step is mainly targeted to visually remove from the diagram (not from the metamodel occurrence) the unused **Measures**, **Dimensions**, and **Descriptors**, or to prune aggregations (**Bases**).


4 The Data Warehousing Flows Visual Library

Data cube diagrams visualise the static aspects of f^w 's. In this section, we discuss how to manage the dynamic aspects. According to the well-known software-engineering principle of *separation of concerns* [9], the dynamic and static features of f^w 's are represented in separate diagrams. By setting aside the complexity of data cubes, f^w 's can be visualised in a more readable form. However, we recall that both kinds of diagrams are closely related, and they were devised as artefacts to be used together, as described in §2.

To represent the dynamic aspects of f^w 's, we could use any type of diagram aimed at process modelling, such as flow charts, data flows, etc. We selected activity diagrams because they enable designers to model f^w dynamics intuitively by means of UML. In this way, both static and dynamic diagrams discussed in this paper may be smoothly integrated to be managed together. Like for data cubes, the many kinds of activities involved in an f^w require an additional customisation. Therefore, we propose (i) a UML profile for adapting the activity diagram notation to represent data cubes, and (ii) a set of f^w catalogues that

capture the f^w diversity. Due to the space constraints, we shall only discuss the OLAP catalogue of the f^w library³. However, all of them share the same principles.

4.1 A UML Profile for Diagramming the f^w Library

As for data cube diagrams, we also devised a UML profile to adapt the notation of activity diagrams. The main issues this profile deals with are: (i) notational improvement, emphasising *object flows* ([3], p.386) for data cubes, and (ii) validation of *action* names that model f^w 's ([3], p.311). As to the first issue, the notation decorates the edges that connect actions by highlighting data cube flows (with the  icon) as shown in Fig. 4. These flows are related to data cube diagrams. As to the second issue, the naming patterns used for actions express the semantics of actions and of their parameters. In such a way, naming patterns can be checked using regular expressions codified by OCL constraints. Due to space constraints, we overlook the details of the profile definition. All the same, the library that applies this profile for visualising f^w 's is described next.

The OLAP catalogue is the main entity of the f^w library due to the relevance of OLAP applications. This catalogue includes the best-known operators of OLAP algebras [10]. With a few exceptions, all operators are f^c . Note that the f^w library contains the best practices in data warehouse modelling, and it is not limited to the presented operators. For instance, some modellers may believe that the *pivoting* operator is relevant enough to be added to the f^w library.

Fig. 4 shows that each OLAP operator is modelled as an action taking cubes in input and output. Naming patterns formalise the vocabulary widely understood among OLAP analysts [10]. Each naming pattern may encode several parameters, represented as `<parameter>`. Optional parameters are enclosed in square brackets. Parameters with multiple occurrences are followed by a “+” mark; in this case, occurrences are separated by commas. Parameters commonly refer to cube elements, and they are instantiated with the same name of the element they refer to. Next, we briefly discuss how the OLAP algebra is rendered; for a deep understanding of this topic, interested readers may be referred to [10]:

slice by has a **criterion** (*i.e.*, a constraint) for filtering values of cube members. This constraint may be given in natural language, *e.g.*, **this year**, or in a formal language such as OCL, *e.g.*, **year = now.year**.

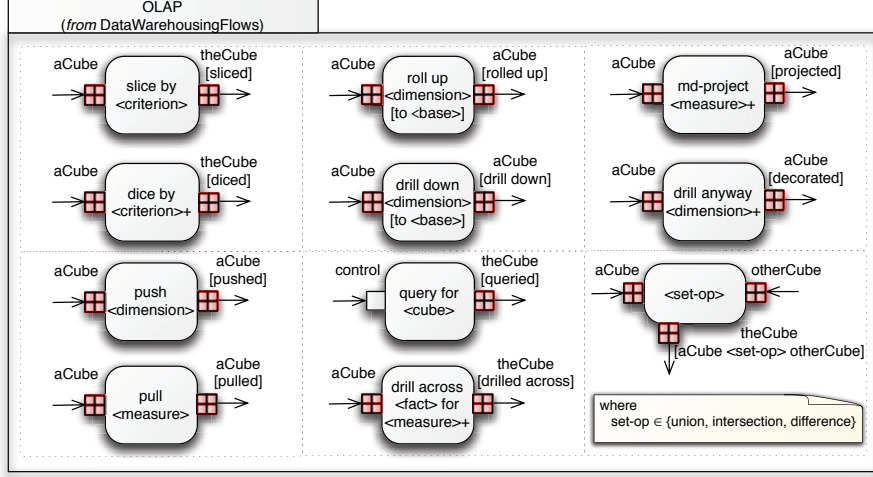
dice by is similar to **slice**, but applied to several dimensions at the same time.

roll up & drill down aggregate and disaggregate data cubes. They are parameterised with a **dimension** and optionally a **base** when multiple aggregation paths are possible from the current aggregation level (*e.g.*, sales by day could be aggregated into months but also weeks).

md-project selects one or more data cube **measures**.

drill anywhere groups cells by a set of **dimensions**. This OLAP operation, also known as *change base*, generalises the **add & remove dimension** operators, that

³ Herein, the term ‘library’ refers to the whole metamodel for the f^w dynamic modelling, whereas each part of this is called ‘catalogue’.

Fig. 4. The f^w catalogue of OLAP actions

can be simulated by using an extended notation: $[+-]\text{dimension}$ for adding or removing dimension to/from the current cube base.

push converts a **dimension** into a **measure**.

pull converts a **measure** into a **dimension**.

query for queries a data warehouse in order to retrieve a data **cube**. This is the only operation that takes a control flow rather than a cube in input. The **cube** parameter is bound to a cube diagram.

drill across joins a **cube** with a **fact** to change the set of **measures**.

union, intersection, & difference manipulate data cubes by using set semantics, thus they are the only actions that require two data cubes in input.

In addition, other set operations such as symmetric difference may be defined by means of the previous ones.

5 Prototype and Example Application

Our framework has been implemented in the ECLIPSE development platform⁴, whose modular, plugin-based architecture enables a proper implementation of the UML extensions proposed. In particular, we have enhanced the plugin for UML modelling, UML2TOOLS, with the functionalities of diagram rendering and event controller for both data cube diagrams and f^w library (see Fig. 5). The proposed models are stored in two kinds of files: those that contain the diagramming metadata and those that contain the modelled elements.

We consider as a case study an OLAP analysis of profitability in the food market domain. The screenshot in Fig. 5 shows in its upper part the whole f^c modelled as an activity diagram: naming patterns are used to name actions, and

⁴ <http://www.eclipse.org> (UML2TOOLS are also located here)

Then it is further decomposed into two sub-branches, the first (f_{2a}^c) performing a **drill across inventory for stock** to traverse interesting data cubes, the second (f_{2b}^c) adding the customer dimension (**drill anyway +customer**). Finally, the two output data cubes c_1 and c_2 for this f^c may be denoted as $c_1 = (f_1^c \circ f_0^c(c_0)) \cup (f_{2b}^c \circ f_2^c \circ f_0^c(c_0))$; $c_2 = f_{2a}^c \circ f_2^c \circ f_0^c(c_0)$. Note that, though the f^w library is extensively used in this example, additional utility actions—such as **duplicate cube**—have also been modelled.

6 Literature Review

There are many modelling frameworks presented in the scientific literature regarding the particular f^w 's. Specifically, concerning OLAP, there are a few works proposing visual modelling techniques for OLAP. [11] presents a specific approach where OLAP sessions are represented by UML state diagrams. Regarding OLAP query modelling, [12] employs a graph-based representation to highlight the f^w dependencies from analytical tools to the data sources. While [12] is oriented to visual modelling of dynamic aspects, [13] achieves static representation of queries by annotating structural conceptual models of a data warehouse, similarly to the data cube diagrams we propose in §3. A compact representation of OLAP queries is achieved by means of UML class-like structures also in [11].

Overall, even OLAP works point out the dichotomy between visualising f^w 's as data states *vs.* data transformations. This is related to the classical debate on *state vs. flow charts*: they are complementary and emphasising one aspect rather than another. It is also related to the dilemma of visualising *structural vs. dynamic* aspects of f^w 's. Citing [14], “every notation highlights some kinds of information at the expense of obscuring other kinds”.

7 Conclusions

The state-of-the-art for visual modelling of f^w 's comprises a wide range of techniques, each taking into account specific aspects of application domains, but overlooking their common foundational concepts. In this work we identified two challenging issues concerning design-oriented f^w visual modelling: how to handle complex data structures such as data cubes, and how to specify the semantics of the involved data transformations in a formal and straightforward mode. For this reason, we devised an f^w visual modelling framework where two kinds of diagrams are provided by using UML as scaffolding. Their suitability to visually manage the complexity involved in f^w 's is shown by applying them to an example scenario relying on the **Eclipse** platform.

The results of this work have interesting implications for data warehouse practitioners. Regarding the integrated vision of f^w 's, the current modelling tools, that were conceived for a specific kind of f^w , may be reused for the others. This fact sets a bridge between current visual modelling techniques. Thanks to the unifying definition of f^w 's, we presented a general framework for their

modelling. This proposal is aligned with model-driven technologies [1] such as those presented in [15] for designing data warehouses.

Some challenging new research topics appear next. Two of them are specially encouraged: automatic code generation from these diagrams by applying model transformations (according to [15]), and the empirical validation of their enhancement in cognitive issues such as readability.

References

1. Favre, J.: Towards a Basic Theory to Model Model Driven Engineering. In: 3rd Workshop in Software Model Engineering, WiSME (2004)
2. Romero, O., Abelló, A.: A Survey of Multidimensional Modeling Methodologies. *International Journal of Data Warehousing & Mining* 5(2), 1–23 (2009)
3. OMG: Unified Modeling Language (UML) Superstructure, version 2.1.2. (November 2007), <http://www.omg.org/technology/documents/formal/uml.htm>
4. Moody, D.: What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development. In: *Adv. in Inform. Syst. Dev.; New Methods and Pract. for the Networked Soc.*, pp. 481–492 (2007)
5. Pirolli, P.: *Information Foraging Theory: Adaptive Interaction with Information*. Oxford University Press, USA (2007)
6. Luján-Mora, S., Trujillo, J., Song, I.-Y.: A UML profile for multidimensional modeling in data warehouses. *Data Knowl. Eng.* 59(3), 725–769 (2006)
7. Object Management Group: Object Constraint Language (OCL), version 2.0. (October 2003), <http://www.omg.org/technology/documents/formal/ocl.htm>
8. Ware, C.: *Information Visualization: Perception for Design*, 2nd edn. Morgan Kaufmann, San Francisco (2004)
9. Tarr, P.L., Ossher, H., Harrison, W.H., Sutton Jr., S.M.: *N Degrees of Separation: Multi-Dimensional Separation of Concerns*. In: *Proc. ICSE*, pp. 107–119 (1999)
10. Romero, O., Abelló, A.: On the need of a reference algebra for OLAP. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) *DaWaK 2007*. LNCS, vol. 4654, pp. 99–110. Springer, Heidelberg (2007)
11. Trujillo, J., Luján-Mora, S., Song, I.-Y.: Applying UML For Designing Multidimensional Databases And OLAP Applications. In: *Advanced Topics in Database Research*, vol. 2, pp. 13–36 (2003)
12. Papastefanatos, G., Vassiliadis, P., Simitsis, A., Vassiliou, Y.: Design Metrics for Data Warehouse Evolution. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) *ER 2008*. LNCS, vol. 5231, pp. 440–454. Springer, Heidelberg (2008)
13. Cabibbo, L., Torlone, R.: From a Procedural to a Visual Query Language for OLAP. In: *Proc. SSDBM*, pp. 74–83 (1998)
14. Green, T.R.G., Petre, M.: Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework. *J. Vis. Lang. Comput.* 7(2), 131–174 (1996)
15. Mazón, J.-N., Trujillo, J.: An MDA approach for the development of data warehouses. *Decis. Support Syst.* 45(1), 41–58 (2008)