A product-line approach to database reporting

Felipe I. Anfurrutia, Oscar Díaz, Salvador Trujillo

ONEKIN Group. University of the Basque Country

PO Box: 649

20009 San Sebastián

Phone: +34 943 018 064

{felipe.anfurrutia, oscar.diaz, struji}@ehu.es

Abstract

Reporting for analysis is a common demand to database programmers. Unlike dynamic data analysis, reporting tend to be foreseeable. Despite this fact, staff waste their time programming similar reports time and again, where reports are often built from scratch with little if any reuse at all. Based on the predictability and similarity among reports, this work presents a product-line approach to database reporting. The feature model, core assets and production plan of the product line are sketched, and the architecture discussed. The work was conducted under a main requirement: data warehouse technology was not available. Either the cost or the lack of appropriate staff makes small-and-medium companies reluctant to use these sophisticated tools while simple reporting is all they need. In this scenario, the product-line approach can be a cost-effective solution to achieve reuse.

1 Introduction

Many small organizations often access data for reporting directly from on-line transaction processing (OLTP) systems. This approach can suffice if sophisticated data analysis is not required, or resources are not available. Data warehouse technology offers effective support for analysis but they also imply qualified staff and equipment that small organizations can not always afford.

Accessing the OLTP system directly for reporting incurs in two main limitations (apart from handling of historical data and ETL issues). First, the system performance for transactions is adversely affected as now the very same data is used to feed both OLTP and reporting applications.

An additional complexity associated with reporting directly from the source system is that the organization of the data is rarely intuitive to a business user. The data has been stored to optimize data entry, not to optimize data access. This often leads to devote specialized personnel to producing reports for others in the organization. These team members are increasingly relied upon to provide new and modified reports, preventing them from completing their assigned duties or forcing a department to dedicate staff to producing reports [12].

Unlike dynamic data analysis, reporting tend to be foreseeable. Based on the predictability and similarity among reports, this work presents a product-line approach to database reporting. Software Product Lines (SPLs) are defined as "*a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*" [6].

A report family is then a set of reports that share some commonalities. The product-line designer should strive to apprehend this commonality as well as identifying their differences. A given report is then a product of this line where a report is characterized by the data used, its layout and its format.

The requirement of the system follows:

1. no data warehouse technology is available. Either the cost or the lack of appropriate staff makes small-and-medium companies reluctant to use these sophisticated tools while simple

reporting is all they need.

2. detaching as much as possible reporting from the database itself. To this end, cubes are extracted and saved as XML files. Then, cubes are conceived as a kind of data cache [1]. A *elapsedTime* feature is added. This feature capture the maximum elapsed time. Based on our experience, this elapsed time tend to be larger enough to allow to re-calculate the cube in batch mode (i.e. normally this implies to run the query at night). Note that in small organizations the volume of data is small enough to make this viable.

3. promoting self-service reporting for end users.

Implementation wise, the system makes intensive use of XML technologies. All, the data cube (*XCube* [13]), the feature model (XML-based [3]), the formatter (*XSL* [17]) and the build process (*Ant* [9, 16]) are all described using XML vocabularies. This makes the system platform-independent, and provides a good example of the increasing usefulness of XML.

The rest of the paper is structured as follows. Section 2 presents a sample case. Section 3 outlines product lines. Section 4 and 5 introduce the feature model and the core assets of the reporting product line, respectively. Finally, section 6 discusses and section 7 concludes the paper.

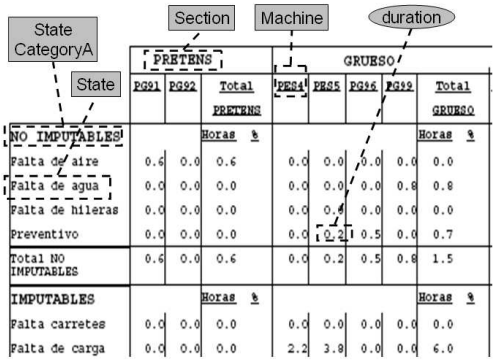

Figure 1: A product-manufacturing report sample for a Spanish customer

2 The sample case

This work has been conducted for the TRACELIA system which addresses the management of the information flows in manufacturing plants. Aspects of the production process to be controlled include production-process monitoring, machine-level inventory control or maintenance control. An example of an analytical query in this scenario is: "*give me the total elapsed time per state for machines of all sections by state category during march 2005*" where "state" refers to whether the machine was idle, out of order and the like. Figure 1 shows a sample report, where states are labeled as *"lack of air"* (*"Falta de aire"*), *"lack of water"* (*"Falta de agua"*), etc.

The analysis requires first to identify the subject of the analysis: the **fact**. In our case, this corresponds to *machine-transition event*s, i.e. entries

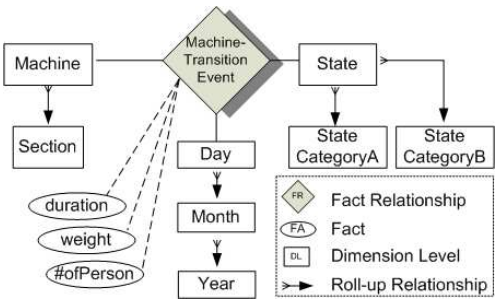

Figure 2: A ME/R model for the data cube

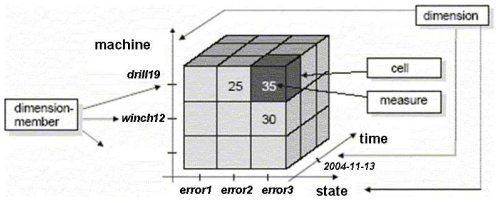

Figure 3: Cube metaphor

submitted by controllers where the machine leaves a state and notifies the time spent in the salient state. This fact is described through **measures** (quantifying data). For our sample problem, the measures include the elapsed time in the state, the weight or the number of persons while in this state. Figure 2 illustrates the conceptual model for this example using the ME/R diagram [15]. This notion depicts the measures as attributes of the **fact relationship** (depicted as a big diamond in figure 2).

These facts can then be analyzed along the following dimensions: time, the machine and the state category. Dimensions allow for distinct degrees of granularities in the analysis. This classification hierarchies are used for the structuring of dimensions along **levels** (e.g. year, month, day). Different levels correspond to different data granularities (e.g. daily figures vs. monthly figures) and ways of classification (e.g. distribution of machine state along time vs. distribution of machine by state). Level A rolls up to a level B if a classification of the elements of A according to the elements of B is semantically meaningful to the application (e.g. the level *days* rolls up to *month*; machine in state *error1* rolls up to state *outOfOrder*).

ME/R uses the **roll-up relationship** to link the distinct granularity levels for a given dimension. Each dimension is represented by a subgraph that starts at the corresponding finest level (e.g. *day* for the time dimension).

The multidimensional design sketched in previous paragraphs, sets the analysis space. The report in figure 1 shows a concrete case where the measure *duration* is analyzed along the *state* and *machine* dimensions. Often a **cube metaphor** is used to represent this data view as shown in figure 3. In the metaphor, axes stands for dimensions whereas cells contain the measures.

Hence, for the point of view of this work, a report can be described as the rendering (i.e. format and layout properties) of a given data cube along a set of dimensions , i.e.

> **report = data cube + dimension + rendering**

This work aims at a self-service reporting system, using product-line techniques to attain this objective.

3 A brief on software product lines

A Software Product Line is "*a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*" [6]. Software product lines give you economies of scope, which means that you take economic advantage of the fact that many of your products are very similar not by accident, but because you planned in that way.

The differences among possible products in the product line can be discussed in terms of **features**. A feature is a "*product characteristic that is used in distinguishing products within a family of related products*" [2]. Features are organized into **feature groups**, which in turn conform a composition hierarchy. Consequently, a software product line must support variability for those features that tend to differ from product to product.

Features are a main input to core asset development. Feature realization mandates variability support for core assets. However, core-asset construction should not be confused with product development. A **production plan** is "*a description of how core assets are to be used to develop a product in a product line*" [4]. For our sample case, this plan includes:

- selection of the features of the desired product, which has to be manufactured,
- realization of the variability by refining core assets to accomplish selected features,
- construction of the product by executing a scripting (realized as the *build.xml* file), which is depicted in figure 6 and described in section 5.3. Thus, final report product is obtained.

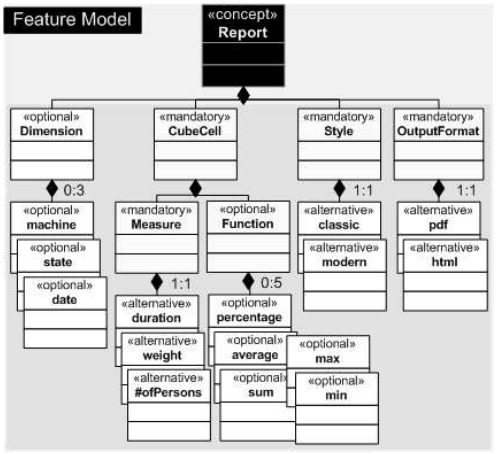

Figure 4: A feature model for database reporting using [5] notation.

Next sections outlines the feature model, the main core assets and the production process of the reporting product line **(RPL)**.

4 RPL: the feature model

A **feature model** provides an abstract and concise syntax for expressing commonality and variability in a specific domain (e.g. database reporting). To attain this, a reuse-oriented analysis of the domain is conducted *[8, 11]*.

The feature model for RPL is depicted in figure 4[1]. Features are annotated with **cardinalities** [14] (e.g. [1..*], [2..2][2]). For database reporting the following feature groups are identified:

- **Analytical dimension**. It specifies the analytical dimensions involved for the sample problem. This group is obtained from the multidimensional model (see figure 2) where the dimension's finest levels correspond to features[3].

- **CubeCell**. This feature includes both the measure to be obtained (e.g. *duration*, *numberOfPersons*, *weight*) and the aggregated function to be used (e.g. *sum*, *min*, *average*, etc).

- **Style**. It establishes the *look-and-feel* of the report. It is similar to the functionality offered by Excel to render the data of a spreadsheet. Alternatives include: *classic* and *modern*.

- **OutputFormat**. It indicates the format of delivery. Alternatives include: *pdf* and *xhtml*.

5 RPL: the core assets

A **Core Asset** is "*an artifact or resource that is used in the production of more than one product in a software product line*" [6]. A core asset may be an architecture, a software component, a process, a document, or any other useful result of building a system. Next, the main RPL core assets are introduced.

5.1 The *dataCube manager* asset

This approach does not rely on the existence of a proper data warehouse. Rather analysis is conducted directly from OLTP data. This can jeopardize the efficiency of operational transactions since they must compete with analysis queries. Moreover, OLTP databases are design to optimize OLTP transactions where tables are designed and termed focusing on effective programming rather than facilitating analysis.

These observations vindicate the need of a kind of "materialized view" where data is engineered for analysis. Due to both the lack of this functionality in some Database Management Systems (DMBSs), and to decouple the DBMS from the RPL, a core asset is defined for materialized-view support: the *dataCube* asset.

The *dataCube* asset includes a definition of the multidimensional schema, a proxy to the DBMS, and the data cube itself which contains the data. To achieve interoperability, the *XCube*[4] proposal was used. *XCube* [13] is an open, manufacturer-independent and XML-based family of document templates to store, exchange and query data warehouse data. This proposal includes *XCubeSchema*

[1] For a complete account on feature modeling refer to [7, 14, 10]. [3] gives an XML vocabulary for feature specifi cation.

[2] Mandatory and optional features can be considered special cases of features with the cardinalities [1..1] and [0..1], respectively.

[3] If no dimension is selected, the report is reduced to a single value (e.g. total duration regardless of the machine, time and state category). If one is selected, the report is a row of values. If more that one is selected, the report turns into multi-dimensional.

[4] http://www.xcube-open.org/

Figure 5: The fact data of the *dataCube* using *XCubeFacts* vocabulary

& *XDimension* for multidimensional schema description (see figure 2), and *XCubeFacts* to describe the fact data. Figure 5 shows an *XCubeFacts* file for the data cube of our sample problem.

One disadvantage of using XML for exchanging data cubes is the fact that XML documents tend to be rather large. But with constantly growing network bandwidth, being in a local setting, and the use of compression methods in mind the advantages prevail. This advantage include interoperability and the existence of a broad range of tools for XML processing.

5.2 The formatter stylesheet

The *formatter.xsl* asset is in charge of both defining a page layout and giving a style format to the data. Thus, this asset is used for building the first outline of the report called *pre-report.fo* document. The *pre-report.fo* document is described using the known XSL-FO[5] vocabulary. This vocabulary is a device independent format, which helps to carry forward the variability on the output format of the report.

5.3 The script

Figure 6 illustrates how the build process is realized by several steps, which are necessary in order to create a single report:

- **Extractor**. A query (*query.cql*) is performed on the cached cube (*data.xcb*) in order to gather only the required data (*data.xml*), depending on the features that identifies a product.
- **Formatter**. This step transforms the obtained *dataCube* (*data.xml*) in a first outline *(pre-report.fo)*, applying the *formatter.xsl* document for that.
- **Renderer**. This outline (*pre-report.fo*) is rendered using selected output format.

The build process is represented by an *Ant* document. *Ant* is a Java-based tool for scripting build processes [9, 16]. Scripts are specified using XML syntax, and often named *build.xml*.

6 Discussion

A product is distinguished by the set of features it realizes. For instance, the report shown in figure 1 can be characterized as being a *PDF* report using a *classic* rendering of the *total duration* fact set up by two dimensions, namely, *state* (their categories) and *machine* (*machineId* and *location*). The number of the distinct products (i.e. reports) that a product line is capable of producing can be obtained as the multiplication of the distinct compatible features of the product line. In our example, the RPL can generate more than one hundred distinct reports.

The important point to note is that new reports are no built from scratch but from a common set of core assets. Table 1 summarizes which core assets affect which features. In this case, the impact can imply to change a configuration parameter (e.g. the aggregate *function* to be used), to add a new plug-in (for the *output* feature) or to extend the XSLT templates (for the style *feature*). Notice that the product process itself can need to be changed to accommodate some features.

However, the question is what if the end user wants a report outside the product-line realm, i.e. with a feature not included in the feature model? It should be said that the domain analysis conducted as part of the product-line development aims at foreseen the distinct requirements from which only a

[5] http://www.w3.org/TR/xsl/

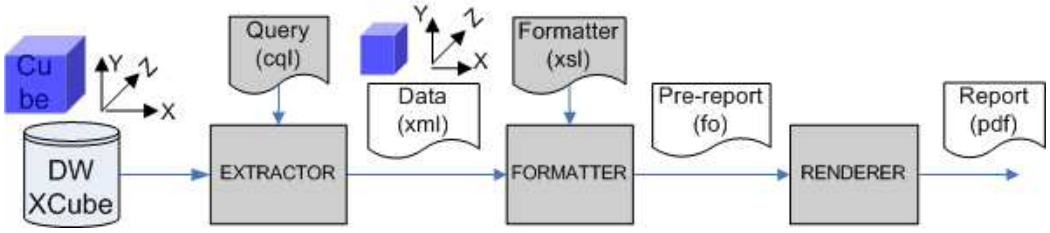

Figure 6: A build process for database report

	dataCube	formatter	script
Dimension			
machine	X	X	
state	X	X	
date	X	X	
Function	X	X	
Fact	X	X	
duration	X	X	
#ofPersons	X	X	
weight	X	X	
Style			
classic		X	
modern		X	
Output			
pdf			X
html			X

Table 1: Features vs Core Assets

subset are selected to be finally supported by the product line (e.g. due to strategic or cost effective reasons). Even though, product lines could need to be revised, although wishfully at a distinct pace than traditional software. But even in this case, the separation of concerns that characterized product-line architectures facilitates extensions.

This project also makes an intensive use of XML throughout all the development of the product line. This accounts for a cohesive framework: all, core assets, the production plan, and the feature model itself are described using XML vocabularies. In this case, the use of a single technology speed up the construction of the product-line framework.

An interesting twist, no investigated yet, is the storage of the product-line artifacts. Being XML, these artifacts are liable to be stored in the database. Specifically, our implementation uses Oracle which support the *XMLType*. A RPL table is defined which holds a set of *XMLType* attributes that describe the distinct roles that the kept document plays. Being a database, SQL can be used to administer the RPL. For instance, SELECT can be used to retrieve whether a given report can be generated whereas GRANT facilities can be used to handle access control among programmers of the reporting line. Rather than the data itself, now a database is used as a repository of the artifacts of the reporting line.

7 Conclusions

Product lines offer a cost-effective solution to artifact reuse. This paper shows a case for database reporting. The system was built under three main requirements, namely, no use of data warehouse technology, detaching as much as possible reporting from the database itself, and promoting self-service reporting for end users.

The system has been fully implemented. Database programmers already enjoy the reuse gains brought by the product-line approach. Report reuse has greatly been improved. Unfortunately, there is not yet any feedback from end users at the time of this writing that could back the usefulness of this approach from their perspective, and to which extend has their autonomy from technical staff being fulfilled.

Acknowledgements

This work was partially supported by the Spanish Science and Education Ministry (MEC) under contract TIC2002-01442. Salvador Trujillo enjoys

a doctoral grant for the MEC. Our gratitude to Rosa Dunis and Unai Bergara from Mondragón Sistemas de Información, Soc. Coop. for sharing with us their insights about reporting in SME, and providing a case for *Tracelia*.

References

[1] J. Albrecht, A. Bauer, O. Deyerling, H. Günzel, W. Hümmer, W. Lehner, and L. Schlesinger. Management of multidimensional aggregates for efficient online analytical processing. In *IDEAS '99: Proceedings of the 1999 International Symposium on Database Engineering & Applications*, page 156, Washington, DC, USA, 1999. IEEE Computer Society.

[2] D. Batory, J.Neal Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, June 2004.

[3] V. Cechticky, A. Pasetti, O. Rohlik, and W. Schaufelberger. Xml-based feature modelling. In *Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004, Madrid, Spain, July 5-9, 2009. Proceedings*, volume 3107 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2004.

[4] G. Chastek and J.D. McGregor. Guidelines for Developing a Product Line Production Plan. Technical report, CMU/SEI, June 2002. CMU/SEI-2002-TR-06.

[5] M. Clauss. Modelling Variability with UML. In *Proceedings of Young Researchers Workshop GCSE'01, the Third International Symposium on Generative and Component-Based Software Engineering*, Enfurt,Germany, September 2001.

[6] P. Clements and L.M. Northrop. *Software Product Lines - Practices and Patterns*. Addison-Wesley, 2001.

[7] K. Czarnecki and U. Eisenecker. *Generative Programming*. Addison-Wesley, 2000.

[8] K. Kang et Al. Feature Oriented Domain Analysis Feasability Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[9] Apache Software Foundation. Apache Ant. http://www.ant.apache.org/.

[10] H. Gomaa. *Designing Software Product Lines with UML*. Addison-Wesley, 2004.

[11] Martin L. Griss. Implementing Product-Line Features with Component Reuse. In *Proceedings of the Sixth International Conference on Software Reuse*, pages 137–152, Vienna, Austria, June 2000.

[12] A. Grohe. Reporting Architectures, 2002. "http://www.dmreview.com /article_sub.cfm?articleId=4903".

[13] W. Hümmer, A. Bauer, and G. Harde. Xcube: Xml for data warehouses. In *DOLAP 2003, ACM Sixth International Workshop on Data Warehousing and OLAP, New Orleans, Louisiana, USA, November 7, 2003, Proceedings*, pages 33–40. ACM, 2003.

[14] K.Czarnecki, S.Helsen, and U.W.Eisenecker. Staged Configuration Using Feature Models. In *Software Product Lines, Third International Conference on Software Product Lines, SPLC 2004*, pages 266–283, 2004.

[15] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the E/R Model for the Multidimensional Paradigm. In *Int. Workshop on Data Warehousing and Data Mining (DWDM 98), Singapore*, volume 1552 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 1998.

[16] N. Serrano and I. Ciordia. Ant: Automating the Process of Building Applications. *IEEE Software*, 21(6):89–91, November/December 2004.

[17] W3C. XSL eXtensible Style Language Transformations (XSLT) Working Draft Version 2.0, 2005. http://www.w3.org/TR/xslt20/.