Towards the Conceptual Specification of Statistical Functions with OCL

Jordi Cabot[1], Jose-Norberto Mazón[2], Jesús Pardillo[2], and Juan Trujillo[2]

[1] University of Toronto (Canada), jcabot@cs.toronto.edu
[2] Universidad de Alicante (Spain), {jnmazon,jesuspv,jtrujillo}@dlsi.ua.es

Abstract. Current proposals for designing information systems lack the mechanisms to define statistical functions at the conceptual level. Therefore, queries containing these kind of functions are defined once the rest of the system has already been implemented, which requires much effort and expertise. In this sense, the goal of this paper is to show the benefits of extending the Object Constraint Language (OCL) with a predefined set of statistical functions.

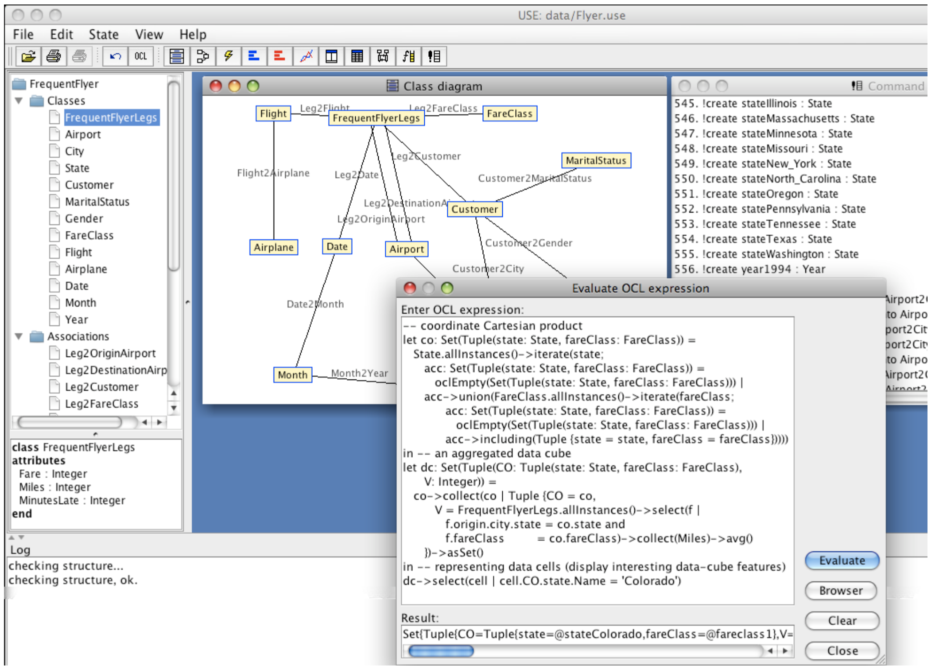
1 Introduction

Queries containing statistical functions are highly important for users to satisfy their information needs in a comprehensive manner [14]. However, information systems design gives little importance to the definition of these kind of complex queries [13]. Currently, queries are not expressed at the conceptual level, thus requiring a lot of effort and expertise in the target implementation platform and preventing designers from validating early on that the conceptual schema satisfies the users' requirements.

The main restriction for defining queries at the conceptual level is the rather limited support offered by current conceptual modeling languages. Surprisingly, essential statistical functions are not predefined in these languages: OCL [12] only provides the *sum*, *size* and *count* operations. ConQuer-II [3] and ConceptBase/TELOS [11] offer just a basic set of predefined statistical functions (as *avg*, *min* and so on). Finally, the ER language [4] does not include a specific language for expressing queries, and complementary query languages proposed later on (as [1, 2, 5]) provide, at most, basic operations.

It is then clear that a better support for statistical functions is necessary to easily express complex queries as part of the definition of a conceptual schema, thus avoiding the error-prone and time-consuming task of defining them once the system is implemented. To this aim, we propose to extend the standard OCL library with a new set of statistical functions that designers can use when defining queries at the conceptual level. These new functions have been tested on sample data by using one of the well-known case studies from Kimball's book [9]: an airline's marketing department wants to analyze the flight activity of each member of its frequent flyer program. The department is interested in seeing what flights the company's frequent flyers take, which planes they travel with, what fare basis they pay, how often they upgrade, and how they earn their frequent flyer miles[3].

The case study has been implemented in the USE tool [7] in order to ensure the well-formedness of OCL expressions and facilitate their validation by providing an evaluation environment. Figure 1 shows the implementation of our case study. In the background of the USE environment we can see the frequent flyers class diagram (left-hand side) and the script that loads the data (objects and links) into the corresponding classes and associations (right-hand side). Given this class diagram, users can request a set of queries to retrieve useful information from the system. For instance, they are probably interested in knowing the *miles earned by a frequent flyer in his/her trips from a given airport (e.g. airports located in Colorado) in a given fare class*. Many other queries can be similarly defined by using other statistical functions in order to analyze data in a richer manner.



```
class FrequentFlyerLegs
attributes
  Fare : Integer
  Miles : Integer
  MinutesLate : Integer
end
```

```
Log
checking structure...
checking structure, ok.
```

```
545. !create stateIllinois : State
546. !create stateMassachusetts : State
547. !create stateMinnesota : State
548. !create stateMissouri : State
549. !create stateNew_York : State
550. !create stateNorth_Carolina : State
551. !create stateOregon : State
552. !create statePennsylvania : State
553. !create stateTennessee : State
554. !create stateTexas : State
555. !create stateWashington : State
556. !create year1994 : Year
```

```
-- coordinate Cartesian product
let co: Set(Tuple(state: State, fareClass: FareClass)) =
  State.allInstances()->iterate(state;
    acc: Set(Tuple(state: State, fareClass: FareClass)) =
      oclEmpty(Set(Tuple(state: State, fareClass: FareClass))) |
    acc->union(FareClass.allInstances()->iterate(fareClass;
        acc: Set(Tuple(state: State, fareClass: FareClass)) =
          oclEmpty(Set(Tuple(state: State, fareClass: FareClass))) |
        acc->including(Tuple {state = state, fareClass = fareClass}))))
in -- an aggregated data cube
let dc: Set(Tuple(CO: Tuple(state: State, fareClass: FareClass),
    V: Integer)) =
  co->collect(co | Tuple {CO = co,
      V = FrequentFlyerLegs.allInstances()->select(f |
        f.origin.city.state = co.state and
        f.fareClass         = co.fareClass)->collect(Miles)->avg()
    })->asSet()
in -- representing data cells (display interesting data-cube features)
dc->select(cell | cell.CO.state.Name = 'Colorado')
```

```
Set{Tuple{CO=Tuple{state=@stateColorado,fareClass=@fareclass1},V=
```

Fig. 1. Conceptual querying of *frequent flyer legs* implemented in USE

Therefore, we believe that it is highly important to be able to provide all kinds of statistical functions as predefined constructs offered by the modeling language so that the definition of complex queries can be carried out at the con-

[3] Note that, in this case study, the interest is in actual flight activity, but not in reservation or ticketing activity.

ceptual level in order to define and validate them regardless the final technology platform chosen to implement the system. This paper is a starting point to address this research, since we show the feasibility of extending the OCL language with statistical functions.

2 Extending OCL with Statistical Functions

Conceptual modeling languages require the use of a general-purpose (textual) sublanguage to express all kinds of queries, constraints and derivation rules since most of them cannot be expressed using only the graphical constructs provided by the conceptual modeling language [6]. For UML conceptual schemas, the Object Constraint Language (OCL [12]) is typically used for this purpose. The goal of this section is to extend the OCL with a new set of predefined statistical functions to facilitate the definition of complex queries on UML schemas.

2.1 Preliminary OCL Concepts

OCL is a rich language that offers predefined mechanisms for retrieving the values of the attributes of an object, for navigating through a set of related objects, for iterating through collection of objects (*e.g.*, by means of the *forAll*, *exist* and *select* iterators) and so forth. As part of the language, a standard library including a predefined set of types and a list of predefined operations that can be applied on those types is also provided. The types can be primitive (*Integer*, *Real*, *Boolean* and *String*) or collection types (*Set*, *Bag*, *OrderedSet* and *Sequence*). Some examples of operations provided for those types are: *and*, *or*, *not* (Boolean), $+$, $-$, $*$, $>$, $<$ (Real and Integer), *union*, *size*, *includes*, *count* and *sum* (Set).

All these constructs can be used in the definition of OCL constraints, derivation rules, queries and pre/post-conditions. In particular, definition of queries follows the template:

```
context Class::Q(p1:T1, ..., pn:Tn): Tresult
body: Query-ocl-expression
```

where the query Q returns the result of evaluating the $Query{-}ocl{-}expression$ by using the arguments passed as parameters in its invocation on an object of the context type $Class$. Apart from the parameters $p1 \ldots pn$, in *query-ocl-expression* designers may use the implicit parameter $self$ (of type $Class$) representing the object on which the operation has been invoked.

As an example, the previous query *total miles earned by a frequent flyer in his/her trips from Colorado in a given fare* can be defined as follows:

```
context Customer::sumMiles(FareClass fc)
body: self.frequentFlyerLegs->select(f | f.fareClass=fc and
          f.origin.city.name='Colorado')->sum()
```

Unfortunately, many other interesting queries cannot be similarly defined since the operators required to define such queries are not part of the standard library. Next, we present our extension to the OCL standard library to include new statistical operators. The set of statistical functions included in our study are those among the most used in data analysis[4]. These functions can be classified in three different groups, following [8, 10]: distributive, algebraic and holistic functions.

2.2 Distributive functions

Distributive functions can be defined by structural recursion, i.e. the input collection can be partitioned into subcollections that can be individually aggregated and combined. One example is the *max* function, which returns the element in a non-empty collection of objects of type T with the highest value. T must support the >= operation. If several elements share the highest value, one of them is randomly selected.

```
context Collection::max():T
pre:  self ->notEmpty()
post: result = self ->any(e | self ->forAll(e2 | e >= e2))
```

2.3 Algebraic functions

Algebraic functions are expressed as finite algebraic expressions over distributive functions, *e.g.*, *average* is computed by using *count* and *sum* functions. The *average* function returns the arithmetic average value of the elements in the non-empty collection. The type of the elements in the collection must support the + and / operations.

```
context Collection::avg():Real
pre:  self ->notEmpty()
post: result = self->sum() / self->size()
```

.

2.4 Holistic functions

Holistic functions are all other functions that are not distributive nor algebraic. For example, the *mode* function, which returns the most frequent value in a collection.

[4] Due to space constraints, we only mention some examples in this paper, but all the defined statistical functions are available in `http://www.lucentia.es/research/ocllib.html`

```
context  Collection::mode(): T
pre:  self ->notEmpty()
post: result = self ->any(e | self ->forAll(e2 |
        self->count(e) >= self->count(e2))
```
.

2.5 Applying the functions

These statistical operations can be used exactly in the same way as any other OCL function. As an example, we show the use of the *avg* function to compute *the average number of miles earned by a customer in each flight leg.*

```
context Customer::avgMilesPerFlightLeg():Real
body: self->frequentFlyerLegs.Miles->avg()
```

In the foreground of Fig. 1 we show one of the queries we have used to test our functions in the USE tool (in this case the query is used to check our *avg* function) together with the resulting collection of data returned by the query. Interested readers can download[5] the scripts and data of our running example together with the definition of our library of statistical functions.

3 Conclusions and Future Work

Support for defining complex queries is very limited in conceptual modeling languages and would hinder designers to directly implement these kind of queries, preventing them from easily satisfying the user requirements. Specifically, queries containing statistical functions cannot be easily defined in OCL since they are not part of the standard library and thus, they must be manually defined by the designer which is an error-prone and time-consuming activity (due to the complexity of some statistical functions).

To solve this problem we argue in this paper that the OCL Standard Library should be extended by predefining a list of new statistical functions that can be used by designers in the definition of their OCL expressions.

Our short term future work is to grow the number of predefined functions in our library and align them with current Model-Driven Development (MDD) and Model-Driven Architecture (MDA) approaches, where the implementation of the system is supposed to be (semi)automatically generated from its high-level models. The definition of all queries at the conceptual level permits a more complete code-generation phase, including the automatic translation of these queries from their initial platform-independent definition to the final (platform-dependent) implementation.

[5] `http://www.lucentia.es/research/ocllib.html`

4 Acknowledgements

Work supported by the projects: TIN2008-00444, ESPIA (TIN2007-67078) from the Spanish Ministry of Education and Science (MEC), QUASIMODO (PAC08-0157-0668) from the Castilla-La Mancha Ministry of Education and Science (Spain), and DEMETER (GVPRE/2008/063) from the Valencia Government (Spain). Jose-Norberto Mazón and Jesús Pardillo are funded by MEC under FPU grants AP2005-1360 and AP2006-00332, respectively. Jordi Cabot is funded by the 2007 BP-A 00128 grant (Catalan Government).

References

1. M. Andries and G. Engels. A hybrid query language for an extended entity-relationship model. *J. Vis. Lang. Comput.*, 7(3):321–352, 1996.
2. M. Angelaccio, T. Catarci, and G. Santucci. QBD*: a graphical query language with recursion. *Software Engineering, IEEE Transactions on*, 16(10):1150–1163, Oct 1990.
3. A. C. Bloesch and T. A. Halpin. Conceptual queries using ConQuer-II. In D. W. Embley and R. C. Goldstein, editors, *ER*, volume 1331 of *Lecture Notes in Computer Science*, pages 113–126. Springer, 1997.
4. P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
5. B. Czejdo, M. Rusinkiewicz, D. Embley, and V. Reddy. A visual query language for an ER data model. *Visual Languages, 1989., IEEE Workshop on*, pages 165–170, Oct 1989.
6. D. Embley, D. Barry, and S. Woodfield. *Object-Oriented Systems Analysis. A Model-Driven Approach.* Youdon Press Computing Series, 1992.
7. M. Gogolla, F. Büttner, and M. Richters. USE: A UML-based specification environment for validating UML and OCL. *Sci. Comput. Program.*, 69(1-3):27–34, 2007.
8. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.
9. R. Kimball and M. Ross. *The Data Warehouse Toolkit.* Wiley & Sons, 2002.
10. H.-J. Lenz and B. Thalheim. OLAP schemata for correct applications. In D. Draheim and G. Weber, editors, *TEAA*, volume 3888 of *Lecture Notes in Computer Science*, pages 99–113. Springer, 2005.
11. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Trans. Inf. Syst.*, 8(4):325–362, 1990.
12. Object Management Group. *UML 2.0 OCL Specification*, 2003.
13. A. Olivé. Conceptual schema-centric development: A grand challenge for information systems research. In O. Pastor and J. F. e Cunha, editors, *CAiSE*, volume 3520 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005.
14. R. B. Ross, V. S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, 2005.