The Constellation Query Language

Clifford Heath

Data Constellation
http://dataconstellation.com/ActiveFacts

Abstract. Collaboration in modeling is essential to success, but ORM diagrams intimidate many business people, and most would never install an ORM modeling tool on their computers.

The Constellation Query Language (CQL) offers an alternative able to represent almost any ORM2 model in plain text using natural language, with the goal of supporting involvement by all parties through familiar tools including email and differential revision management.

The free open source implementation includes robust mapping and code generation for both object-oriented and relational models. Being bootstrapped on a metamodel that is also expressed in the Constellation Query Languge, it forms the basis of a new generation of extensible tools for business requirements management, design and construction of databases and application software, and end-user query facilities.

1 Background

The 1980's adoption of relational databases [1] required new modeling skills and tools, and Object Role Modeling [2] was one approach applied to meet those needs. However, the widespread adoption of object orientation [3] by developers and the difficulties in mapping to relational implementations [4] has become the basis for a deep political divide [5]. Data modelers are often sidelined because they are seen as standing in the way of Agile [6] practises, despite the occurrence of data quality issues their skills could have prevented.

The difficulty of adequately specifying requirements for software systems remains a major cause of software project failure [7] [8] [9]. The linguistic approach adopted by the Object Management Group in the Semantics of Business Rules and Business Vocabulary (SBVR) [10] reflects a focus on desired outcomes, and should help engage the business domain experts, though it makes no concessions to implementation.

The Constellation Query Language follows a similar linguistic approach, but also provides mapping to both object and relational forms, which it is hoped can help bridge both the object-relational chasm and the business specification one.

This paper presents the static aspect of CQL through examples. Queries are shown only in passing due to lack of space. For a full presentation of the grammar with syntax diagrams, please see [11].

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 682–691, 2009.

2 Scope and Goals

In the author's use of ORM2 in a business context, communication with business experts is hampered by the graphical presentation. Although they find the presence of fine detail in a graphical presentation comforting to see, it seems to engender an unwillingness to participate. Context is spread across many diagrams, and structural details like uniqueness constraints are ignored by business people as mere hieroglyphics. Rather than relying on generated verbalisations, the Constellation Query Language (CQL) adopts plain text as the primary input method as a way to make these details more accessable.

Despite adopting a linguistic approach, CQL does not attempt to comprehend unrestricted natural language [12]. It reserves particular verbal expressions, and otherwise uses sequences of arbitrary words (interspersed with *terms* and *quantifiers*) as predicates to designate fact types, which aids understanding.

The name **Constellation** refers to the selection (by a query) of facts from a universe of facts, as we pick out stars in the sky.

Among the goals of CQL are to:

- Include all capabilities of ORM2, using the same formalism,
- Express business requirements clearly and accurately,
- Minimise markup and eschew graphical elements, colour and subscripting,
- Strictly limit the use of reserved words and phrases (most special expressions are context-sensitive rather than reserved),
- Extend naturally to express queries (derived fact types) powerful enough eventually to surpass SQL,
- Being bootstrapped on a clearly-defined metamodel, to support structured requirements management tools,
- Make it easy to produce non-English derivatives,
- Unify relational and object models by emitting both.

CQL is inspired by and resembles ConQuer [13] [14] and the Microsoft ActiveQuery product [15] but avoids the need for a hierarchical structure or graphic elements. Other precursors are FORML[16], RIDL[17] and LISA-D[18]; comparisons have not been undertaken.

CQL differs from the automated verbalisations generated by some ORM tools in that the language is designed to be parsed by a machine without the help of coloured markup.

The implementation was bootstrapped through the use of the Natural Object Role Modeling Architect [19] (NORMA).

3 CQL Statements, Lexical Conventions and Parsing

Each CQL file defines one *vocabulary*, which must be declared first. A vocabulary can incorporate or borrow terms from other vocabularies. The syntax is not shown here. In the examples below, key words and phrases are *in italics*, but this is only for the purpose of clarifying this presentation.

A CQL statement is either a definition terminated by a semi-colon, or a query terminated by a question-mark. A definition defines an object type, a fact type, a constraint, or a unit.

Object types in CQL are designated by single-word *terms*. Title case is not mandatory, but because CQL is case-sensitive and a term may not be used other than to designate roles of the object type, it is normally preferable to maintain this style, so the lowercase form is unrestricted. In future, multi-word terms will be introduced, but this requires further extension of the parsing technology, which also currently limits the use of compound adjectives.

White-space is not significant, and comments are allowed anywhere white-space is allowed (to end-of-line commencing with //, or enclosed in /* ... */ as in C). Comments are not interpreted.

Parsing a CQL statement consists of identifying recognised expressions and terms (with any applicable adjectives) while ignoring the residual predicate text. The combination of the object types (as designated by terms) and the predicate text is used to designate fact types.

Efficient resolution of the ambiguity introduced by allowing arbitrary text is possible due to the adoption of a powerful recent development in parsing technology, Parsing Expression Grammars [20], with lookahead. PEG grammars operate using recursive descent, but provide efficient backtracking through the use of "memoization". Ambiguity is always formally resolved before the end of each definition.

3.1 Value Type Definitions

Value types represent lexical concepts, or categories of things that are represented by literal values. In CQL, they're distinguished by the predicate *is written* (for base types) or more commonly *is written as* (for subtypes). Value types may also be associated with a **unit**, either from a library of known units or defined using the syntax described below under **Unit definitions**.

```
Integer is written;
CompanyName is written as String(60);
Length is written as Decimal(14, 3) in millimeters;
YearNr is written as Integer restricted to {1900..2100};
```

A value restriction contains a list of values and/or ranges. A range may be open-ended. The data type (storage representation) and type parameters (length, scale, etc) are defined by the supertypes.

3.2 Fact Type Definitions

Fact Types are defined by a comma-separated list of fact type readings. The object types that play roles in the fact type must be previously defined. Each reading must include one reference to each role, interspersed in a sequence of predicate words. There are only a few restrictions on predicate words; no term

may be used (that would add a role), and words like *and, if, only, or, but, no, none, maybe*, that form logical expressions, and phrases that introduce quantifiers (see below) and value restrictions are disallowed.

```
Company is called one CompanyName;
Person is called given-Name, given Name is of Person;
Person was born at at most one birth-Place;
Driver drove Vehicle,
    Vehicle was driven by at most one Person (as Driver);
```

The **quantifiers** *one* and *at most one* here create mandatory and/or uniqueness constraints, and are not part of the reading. See below for more details.

The second example shows how a term may be used with a leading adjective. In CQL, adjectives introduce a compound name to refer to that role within this fact type definition, and later when the fact type is invoked in other statements. A leading adjective is indicated by an immediately following hyphen in at least one reading in the list. Trailing adjectives are also supported for languages that require them, immediately preceded by a hyphen. The adjectives must be used in all other cases where that role appears. It is not always required (as in ORM's hyphen binding) that the hyphens have an adjacent space on the other side. This makes it illegal to use hyphenated words like semi-trailer, though this restriction may later be removed if neither word is a term.

The third example shows that although the phrase *at most one* is reserved in this context, the word *at* is not a keyword.

An alternate means of role reference is to use a role name. The final example above defines `Driver` as a name for the role of Person. The definition is in parentheses preceded by *as*. A role name may be defined in any reading of the list, but must be used to refer to that role in all other readings in this definition. The name is purely local - a later invocation must say `Person drove Vehicle`, not `Driver drove Vehicle`. In this respect, CQL is aligned with ORM2 rather than SBVR which has first-class role names, but this is subject to discussion.

3.3 Entity Type Definitions

Entity types represent non-lexical concepts. An entity has no written value, but is instead identified by one or more roles it plays in fact types. In an entity definition, the fact types must also be defined, either using a comma-separated list of fact type **readings** introduced by *where*, or using a reference-mode shorthand indicated by the keyword *its*.

The logical need for identification of entities is enshrined in the CQL syntax that defines them, using *is identified by*:

```
Person is identified by given-Name and family-Name where
    Person is called one given-Name,
    family-Name is of Person, Person has one family-Name;
```

```
Company is identified by its Name(60);

Object is identified by parent-Namespace and Name where
    Object is contained in at most one parent-Namespace,
    Object has one Name;
Namespace is a kind of Object;
```

The first example defines an entity type `Person` whose identifying roles `given-Name` and `family-Name` are both played by `Name`, according to fact types defined in the *where* section.

The second example uses the **reference mode** shorthand to assert a value type called `CompanyName` of length 60 as a subtype of `Name` if not already defined. One, both or neither may have been previously defined. This syntax provides the identifying fact type with default fact type readings `"Company has one CompanyName", "CompanyName is of at most one Company"` if the definition does not include a *where* clause that gives an alternate reading.

The final example shows that an identifying role may be forward-referenced from the *is identified by* section. This is the only case where forward references are allowed between CQL statements, and the only case where it is necessary.

3.4 Entity Subtype Definitions

A subtype is an entity type which has one or more supertypes. Unless otherwise defined, they are identified through the role played by the first supertype.

```
Woman is a kind of Person;
Employee is a kind of Person identified by its Number;
FemaleEmployee is a kind of Employee, Woman;
```

As in ORM2, subtypes are not exclusive unless so constrained, and the subtyping fact type can be invoked using either the predicate *is a kind of* or *is a subtype of*. Possibly in future the reverse predicate *is a/an* will also be provided, though in English that predicate can be misleading.

3.5 Objectification

Objectified fact types must always be named by a term. They are usually introduced by the phrase *is where* before the fact type definition. The second example here shows that objectification also occurs when a normal entity type defines a fact type in which it does not play a role (usually where the fact type has non-standard identification):

```
Directorship is where Person (as Director) directs Company;
TypeInheritance is identified by its ID where
    sub-Type inherits Type;
```

In the Directorship example, the objectified fact type is identified by the implicit uniqueness constraint over (`Person, Company`). This constraint is implied by the absence of an explicit uniqueness constraint.

The first reading determines the role order in the primary key in the relational mapping.

Note that the `given-Name` example in Section 3.2 is illegal unless a uniqueness constraint is somewhere defined over just one role; otherwise it will be objectified, and objectified types must have a term.

3.6 Internal Constraints

Each reading may include one quantifier expression, which precedes the last role in that reading. Quantifiers are used to express simple uniqueness, mandatory and frequency constraints (collectively referred to as **presence** constraints). When a uniqueness constraint is implied by a quantifier, it is equivalent to the ORM constraint spanning all the other roles in the fact type. The requirement in ORM2 that a UC must span either all roles (or all but one) is thus impossible to avoid. The quantifiers are *`at least N, at most N, one, exactly one, at least N and at most M, from {N..M}`*, where N or M is any positive integer or the word *one*. Certain other words may appear in place of a quantifier in queries or constraints, including *`no, none, some, that`*.

The mapping to ORM's uniqueness, mandatory and frequency constraints should be obvious, with one additional detail. Where a minimum frequency is implied, that frequency is mandatory. To handle the very infrequent need to preserve a minimum but make it optional, precede the whole reading by the reserved word *maybe*, similarly to ConQuer.

```
maybe Company is directed by at least 2 Person;
```

Role value restrictions may be included in a fact type definition introduced by the keywords *restricted to*:

```
Person is of Age restricted to {0..120};
Entrant competes in Class restricted to {'A'..'F', 'PW'};
```

Ring constraints (which are external in ORM2) are defined by keywords in square brackets following a fact type reading. A more lengthy verbal form is yet to be defined to handle more complex cases with n-ary fact types.

```
Package is inside at most one container-Package [transitive, acyclic];
```

3.7 External Constraints

Most ORM2 external constraints are represented in CQL by definitions that combine lists of fact type readings, or in some cases, lists of readings joined by *and*. Some ORM2 constraint types involve implicit joins, and these are generally made explicit in CQL. Subtyping joins may be left implicit as long as the

meaning is not ambiguous. CQL joins are more powerful than ORM2's implicit joins because they can traverse more than one object type, and so they cover cases where ORM2 requires use of textual notes or fact type derivations. CQL allows multi-way joins in all types of external constraint, using the prefix *maybe* for outer join semantics.

Presence constraints (uniqueness, mandatory and frequency) can often be included into the fact type definitions where there is a reading with the roles in the correct order, but where that is not possible the following syntax is used.

```
each Incident occurs one time in
    Claim concerns Incident;
each combination Series, Number occurs at most one time in
    Race is in Series,
    Race has Number;
each Person occurs at least one time in
    Person played Role in Event according to Source;
each ContactMethods occurs at least one time in
    ContactMethods includes mobile-Phone,
    ContactMethods includes business-Phone,
    ContactMethods includes Email;
```

The first example is a mandatory and uniqueness constraint in one definition. The second is a uniqueness constraint. The third and fourth are both mandatory constraints.

Subset constraints use *only if*, and equality constraints use *if and only if*. Note the join in the second example, which is implicit in ORM2.

```
Student represents School in Activity
    only if School sanctions Activity;
PurchaseOrderItem matches SalesOrderItem
    only if PurchaseOrderItem is for Product
        and SalesOrderItem is for Product;
Race has Number
    if and only if Race is in Series;
```

Set exclusion constraints (inclusive and exclusive) use *for each* before a comma-separated list of role names before *exactly one* or *at most one* quantifiers. An unimplemented feature also supports a more succinct inclusive and exclusive-or between two cases using *either/or* and *either/or... but not both.*

```
for each DispatchItem exactly one of these holds:
    DispatchItem is for TransferRequest,
    DispatchItem is for SalesOrderItem;

either DispatchItem is for TransferRequest
    or DispatchItem is for SalesOrderItem but not both;
```

3.8 Instances

A value instance is defined by stating the name of the value type followed by the lexical representation of the value. An entity having a single identifying role may be defined exactly the same way, which also defines the required identifying instance (in the third example here, a value). Within a vocabulary, an instance is asserted into the sample population, but in other contexts another population may be the target (the metamodel supports arbitrary named populations).

```
CompanyName 'Microsoft';
Year 1999;
Company 'Microsoft';
```

A fact instance is any reading of the fact type with values following all roles. Entity types having more than one identifying role are defined using **joined** (using *and*) fact instances. Adjectives or role names may be used where an object type plays more than one role.

```
Employee 47 works at Company 'Microsoft';
given-Name 'Bill' is of Person and
    Person is called family-Name 'Gates';
```

3.9 Unit Definition

Units are defined by declaring conversion formulæ involving base units. Both singular and plural names may be given. If a base unit is not otherwise defined, it is assumed to be fundamental. Formulæ are limited to a coefficient and an offset ($ax + c$), where x may be any number of existing units each raised to any integer power. Conversion coefficients have a real numerator and an integral denominator which defaults to 1. The formula may be written in either direction, but the constants must be on the side of the base units. An extensive library of standard unit conversions is available. Ephemeral conversions can be defined; the conversion factor at a particular time is assumed to be available from some source. Approximate conversions can be annotated.

```
25.4 millimeters converts to inch/inches;
kelvin + 273.15 converts to celsius;
9/5 celsius + 32 converts to fahrenheit;
acceleration converts to metres second^-2;
g converts to 9.7 acceleration approximately;
0.853 dollarUS converts to dollarAU ephemeral;
```

The use of units in defining value types and in query literals allows dimensional analysis and automatic conversions. This example shows a derived fact type ("Pane has Area") and a query that uses units conversion to list large panes of glass. Notice that the literal in last line has an associated unit (this line also has a **contracted join**, see below).

```
Dimension is written as Real in millimeters;
Width is written as Dimension; Height is written as Dimension;
Pane has one Width; Pane has one Height;
Pane has Area where
    Pane has Width, Pane has Height,
    Area = Width * Height;
Pane has Area > 5 foot^2?
```

3.10 Business Context Notes

Each entity type, value type, and fact type, as well as each CQL constraint and value (or role value) restriction, may display one or more **business context notes**. These allow arbitrary text to be introduced by specified phrases, and for the text to be associated with an author, a date, and a list of those who approved it. The types of context note include *because, as opposed to, so that, to avoid*, and each context note, in parentheses, may be preceded by *according to ...* and followed by *as approved by ....* The only restriction on the included text is that any parentheses must be matched. CQL comments are disabled inside the text. It is expected that this feature will see more use in a structured requirements management tool than in plain text CQL.

```
Person is called one (as opposed to more than one, because we'll
join them into a single string, approved by Joe, Bill) given-Name;
```

3.11 Upcoming Work

A language feature called **join contraction** will allow joins to be expressed more succinctly, using either *who* or *that*. This statement asserts three value instances, two entity type instances and three facts. Can you spot them all?

```
given Name 'Bill' is of Person
    who is called family Name 'Gates' and
    that Person directs Company 'Microsoft';
```

The details of queries and derived fact types are mentioned but not covered in this paper, and are as yet largely unimplemented. They involve **comparison** clauses in addition to invoking fact types as previously shown.

Queries also provide a **returning** clause to delineate which related facts should be available in the result set and how the results should be ordered, so that a query may return more information than just listing the derived fact's role players. Queries can refer to some **contingent facts** such as the current system time and system name, the current user, etc. Uniqueness always applies to the derivation of any fact type, so all capabilities of SQL pertaining to its keywords DISTINCT, GROUP BY, HAVING, etc, are available by using nested queries.

3.12 Conclusion

The free implementation of the Constellation Query Language has already provided a new way to specify and develop robust semantic models for an information system. As the query facilities grow to rival the power of SQL, untrained users will be empowered to help specify, to understand and to interact with data systems in their own language.

References

1. Codd, E.: A Relational Model of Data for Large Shared Data Banks. CACM 13(6) (1970)
2. Object Role Modeling, http://www.ormfoundation.org
3. Unified Modeling Language, http://www.omg.org/technology/documents/formal/uml.htm
4. Ted Neward (2006), http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx
5. Ambler, S.: The Cultural Impedance Mismatch (2009)
6. 17 co-authors (2001), http://agilemanifesto.org/
7. Standish Group: Collaborating on project success (2001), http://www.softwaremag.com/archive/2001feb/collaborativemgt.html
8. Nierstrasz, O., Demeyer, S., p. 30 (2000), http://scg.unibe.ch/archive/lectures/ESE-W00.pdf
9. David, A.: Just Enough Requirements Management (2004), http://conferences.codegear.com/kr/article/32301
10. Object Management Group: The Semantics of Business Vocabulary and Business Rules (2008), http://www.omg.org/spec/SBVR/1.0/
11. Heath, C.: Introduction to the Constellation Query Language (2007-2009), http://dataconstellation.com/ActiveFacts/CQLIntroduction.html
12. Dijkstra, E.W. On the foolishness of natural language programming
13. Bloesch, A., Halpin, T.: ConQuer: a conceptual query language. In: Thalheim, B. (ed.) ER 1996. LNCS, vol. 1157, pp. 121–133. Springer, Heidelberg (1996)
14. Bloesch, A., Halpin, T.: Conceptual queries using ConQuer-II. In: Embley, D.W. (ed.) ER 1997. LNCS, vol. 1331, pp. 113–126. Springer, Heidelberg (1997)
15. The Microsoft ActiveQuery product has been withdrawn from sale
16. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
17. Meersman, R.: The RIDL conceptual language, Research report, Int. Centre for Information Analysis Services, Control Data Belgium, Brussels, Belgium (1982)
18. Hofstede, A.H.M., ter Proper, H.A., van der Weide, P.: Formal definition of a conceptual language for the description and manipulation of information models. Information Systems 18(7), 489–523 (1993)
19. The Natural Object Role Modeling Architect, http://ormfoundation.org/files
20. Ford, B.: Packrat Parsing: a Practical Linear-Time Algorithm with Backtracking. Massachusetts Institute of Technology (2002), http://pdos.csail.mit.edu/baford/packrat/thesis