

Complex and Dynamic Constrains of Semantically Complete Model

Vladimir Ovchinnikov, PhD in Computer Science

Russia, 398024, Lipetsk, prospekt Pobedi 104-44,
ovch@lipetsk.ru,
Lipetsk State Technical University, Russia,
Novolipetsk Iron & Steel Corporation, Russia.

Abstract. In the most general case any constraint represents a predicate defined over queries of two types: a) executed within the current model state, b) executed within the next model state after the current transaction commits. A constraint is dynamic if it is based on both current and next states; otherwise it is based on the current state only and is static. A constraint is complex if it is based on queries more complicated than a natural join (composition); otherwise it is simple. Complex and dynamic constraints are not covered by existing conceptual modeling techniques. The paper discusses principles of conceptual constraint formulation of any type within SCM conceptual modeling technique, using its embedded conceptual query language SCQL. SCM and SCQL are purely based on application domain concepts without any complicated constructions. High transparency and simplicity of SCM and SCQL make it possible to use the approach by technical as well as non-technical persons.

Keywords. conceptual constraint, conceptual modeling, conceptual query language, Semantically Complete Model, Semantically Complete Query Language

1. Introduction

Semantically Complete Model (SCM) [Ov04, Ov05a, Ov05b] can be considered as a restriction of ORM [Ha01] with the following main constraints: a) each association (fact type) is uniquely identified with a set of object types underlying it within a model; b) an association can not be based on an object type set being proper subset of another object type set underlying another association of the same model; c) there can not be two equivalent association instances (having equal object type instance combinations). The first constraint is covered by the second one; the first one is picked out since it is of high importance and it has resulted in creation of SCQL (Semantically Complete Query Language [Ov05a, Ov05b]) over SCM which does not use proper association names for query formulation.

The main property of SCM is semantic completeness which follows from the given constraints and means that within a model each association defines interconnection of underlying object types completely. In other words, knowing a set of object types, we

can reason about semantics of their interconnection completely; it is not necessary to operate with associations as separate entities and to use associations' proper names. The property leads to more transparency of SCM model for technical and non-technical users since they can think in the plane of object types (concepts) merely. SCQL adopts the property and permits query formulation using application domain concepts only.

As a result of this particular property, object types are further called as concepts in the context of SCM. The concepts are considered to be part of an appropriate application domain and to be operated by the domain's experts directly. The paper concerns structure of SCM constraints as a whole and two types of constraints (complex and dynamic) in particular.

SCM has no many ORM particularities, such as objectification [Ha05], differentiation between abstract and concrete object types, alternative associations [Ov04], and others. We believe the aspects belong to the implementation level rather than the abstract level to a greater extent. In SCM approach, the aspects appear at the level of mapping of a SCM model to more implementation specific models (for example, ER [Ch76], ORM, relational schema, UML [OMG03b, OMG03c, OMG04], and others). It results to greater abstracting of SCM from implementation details. Meanwhile, SCM remains sufficient for complete conceptual modeling of any application domain.

According to OMG Model Driven Architecture [OMG03a], the system implementation process can be represented as mapping of a Computational Independent Model (CIM) to Platform Independent Model, and the latter to Platform Specific Model. We consider SCM and ORM as CIM which can be used alternatively. But they can be effectively used in one project at the same time. In this case, we should first define a SCM model and then map (extend) it to an ORM model, adding some implementation details. Only complex and dynamic SCM constraints discussed further can not be mapped to ORM constraints since ORM has no the appropriate mechanisms yet as it provides only constraint templates (macroses) [Ha97].

Main notation of SCM is textual: it is more readable by non-technical persons and it can contain complex and dynamic constraints describing essence of a domain in more details. SCM has its own notation terminology due to its particular properties and completely textual representation. Terminologies of SCM and ORM are rather different as ORM is not adapted to entirely textual view (see the next section). The difference affects graphical notations as well because we tried to approach SCM graphical and textual views. A SCM model can be represented graphically as a hypergraph having some additional expressive means and restricted according to the foresaid constraints; therefore its graphical notation is a kind of extended hypergraph notation: concepts (nodes) are designated with ellipses, associations (edges) – with lines or outlines connecting ellipses, and simple constraints are drawn over. When an association is n-ary, star line and outline can be used interchangeable, for instance, fig. 1 shows two equivalent notations of a ternary concept association.



Fig. 1. Two equivalent notations of the ternary SCM association (Person, Skill, Skill Level)

Star lines are mostly used when some simple constraints are formulated over associations. For instance, the constraint “for each person and skill combination there can be only one skill level” looks as fig. 2 shows.

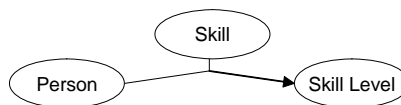


Fig. 2. The ternary SCM association (Person, Skill, Skill Level) constrained with the functional constraint [(Person, Skill) \rightarrow Skill Level]

SCM graphical notation is somewhat simpler than ORM’s one and more compact; it reflects ideas of SCM more precisely, including terminology of constraints. Not all complex constraints can be presented in graphical notation compactly because of their arbitrary structure. Therefore graphical notation of SCM is not complete; it is used for presentation purposes only and it should be accompanied by all missing constraints.

SCM textual notation is a set of association sentences accompanied by constraint expressions. Association sentence corresponds to a concept association and presents it in the verbal form as one sentence describing concepts’ interconnection; concepts are distinguished with capital first letters for all words. For instance, the association (Person, Skill, Skill Level) can be presented as the association sentence “Person has a Skill of a Skill Level”. Constraint expression is a formal expression of the SCM constraint language in square brackets. For instance, [(Person, Skill) \rightarrow Skill Level] is the constraint expression meaning “for each person and skill combination there can be only one skill level”. If a constraint expression concerns only one association, it follows after the association’s sentence (see the running example below). The detailed description of SCM constraints’ syntax is out of the scope of the paper; in the next sections, we will discuss the principles and show some examples.

SCM has the analogous simple constraint types as ORM has; they are considered in the next section. A complex constraint is a constraint based on a query more complex than a path expression. ORM does not permit formulation of such constraints because the technique has no an embedded query language. There are some query languages based on ORM (for instance, LISA-D [HPW93, HPW96], Conquer [BH97]), but they are not adapted for constraint formulation. To the contrary, SCM has the embedded language SCQL which is a declarative conceptual query language not using proper association names for query formulation. Presence of SCQL within SCM modeling technique permits formulating complex and dynamic constraints as it is described in the sections 3, 4.

The table 1 introduces a running example of SCM model of a project management domain aspect. It is used in the next sections for illustration purpose. The SCM model is self-contained since it includes clear association sentences and natural language

verbalization of constraints (using '<>' for static constraints and '<<>>' for dynamic constraints). Here we only emphasize the fact noted above that each association is identified with a concept set underlying it and they are not included one to another.

Table 1. A running example: SCM model of a project management domain aspect

1	Person has a Skill of a Skill Level
2	[(Person, Skill) → Skill Level] <There is only one skill level for each
3	combination of person and skill.>
4	[Skill Level = {"Beginner", "Medium", "Expert"}]
5	Person has <u>Assignments</u> ←
6	<u>Assignment</u> concerns a Task →
7	[(Person–Assignment–Task): Assignment ≡ (Person, Task)] <There is
8	only one assignment for each combination of a person and a task.>
9	<u>Assignment</u> has a Time Amount →
10	<u>Assignment</u> has a Percent Complete →
11	[Percent Complete = [0..100]]
12	[(Percent Complete–Assignment–Time Amount(Assignment)),
13	(SELECT Assignment, SUM(Time Amount)
14	FROM (Assignment–Solution Act–Time Amount)):
15	Percent Complete=Time Amount(Assignment)/SUM(Time Amount)]
16	<Percent complete is a ratio between total time amount of solution acts
17	made for this assignment and its own time amount.>
18	[(^ (Person–Assignment–Task), (Assignment)) is part of (Person–
19	Assignment–Task)] <<Assignment cannot be reassigned to another
20	person or task.>>
21	Person has made <u>Solution Acts</u> ←
22	<u>Solution Act</u> concerns a Task →
23	[(Person–Solution Act–Task): Solution Act ≡ (Person, Task)] <There
24	is only one solution act for each combination of a person and a task.>
25	[(Person–Solution Act–Task).(Person, Task) is part of
26	(Person–Assignment–Task).(Person, Task)] <If there is a solution act
27	made by a person in favor of a task, then the person should be already
28	assigned to the task.>
29	<u>Solution Act</u> corresponds to an Assignment →
30	[(SELECT Solution Act, Assignment FROM (Person–Solution Act–
31	Task–Assignment–Person)) equals (Solution Act, Assignment)] <A
32	solution act corresponds to that assignment which corresponds to the
33	same combination of a person and a task.>
34	[(^ (Solution Act–Assignment), (Solution Act)) is part of (Solution
35	Act–Assignment)] <<Solution act cannot be changed to another
36	assignment.>>
37	<u>Solution Act</u> has Time Amount →

The equivalent graphical notation of the same model is presented in fig. 3. It contains only simple constraints, and complex constraints are to be supplied with it.

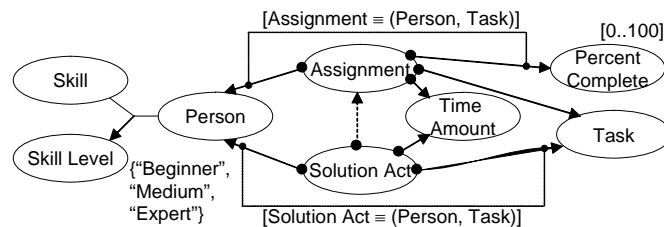


Fig. 3. Graphical notation of SCM model of a project management domain aspect

2. Simple SCM constraints

Constraint template is a parameterized predicate defined over a) instances of an association or a simple query result, b) whole states of several associations or simple queries results. For instance, the functional constraint template ‘ \rightarrow ’ belongs to the first category and means “for all instances of an association (query) it is true that each combination of determining concepts’ instances corresponds to only one combination of depending concepts’ instances”. The functional constraint template is parameterized by two concept sets and one association. Each functional constraint is an instantiation of the template with concrete concept sets and association. In the running example, the constraint [(Person, Skill) \rightarrow Skill Level] is defined for the association (Person, Skill, Skill Level) and is read as “there is only one skill level for each combination of person and skill”. If SCM constraint of the category (a) is based on a query, not an association, it is written down as [\langle query \rangle : \langle simple constraint \rangle]. For instance, the constraint “[(Person–Assignment–Task): Assignment \equiv (Person, Task)]” is based on the SCQL path expression query (Person–Assignment–Task).

Inclusion constraint template “is part of” is an example of the second template category since it is based on two query results in general case (association states in particular). The template means “all instances of the first query result are to be within instances of the second query result”. In our example, an instance of the template is the constraint [(Person–Solution Act–Task).(Person, Task) is part of (Person–Assignment–Task).(Person, Task)] which reads as “if there is a solution act made by a person in favor of a task, then the person should be already assigned to the task”. Syntax and semantics of the used query language are briefly given in the next section.

A constraint is considered to be simple if it is an instance of a constraint template parameterized by associations or simple queries. A query is considered to be simple if it represents a composition of several associations merely. All other constraints are complex and their structure will be discussed in the section 4.

If a constraint template is used within an association sentence, it defines a constraint based on concepts of the association. For instance, “Person has Assignments \leftarrow ” defines the association (Person, Assignment) and two constraints: mandatory “Assignment” and functional “Assignment \rightarrow Person”. The running example uses the following templates for constraint formulation (see table 2). ORM analogues for the SCM constraints are shown at table 2.

Table 2. SCM constraint templates used in the running example

SCM constraint	Textual Notation	Graphical Notation	Meaning	ORM constraint
Mandatory	\underline{x}	•	All existing instances of x participates in the association 's state	Mandatory role
Functional	$x \rightarrow y$	\rightarrow	For all instances of a query it is true that each instance set of x has only instance set of y	Uniqueness (one role)
Equivalence	$x \equiv y$	\equiv	$x \rightarrow y$ and $y \rightarrow x$ over the same query	Uniqueness (two roles)
Instance	$[x..y]$ $\{x, y, \dots\}$	$[x..y]$ $\{x, y, \dots\}$	There can exist only enumerated (spanned) instances of the concept	Value
Equality	x equals y	x equals y	A result of the query x is equal to a result of y	Equality
Inclusion	x is part of y	x is part of y	A result of the query x is part of a result of the query y	Subset

This is not complete list of SCM constraint templates, but the complete list is out of the scope of the paper. All ORM constraints can be formulated using simple SCM constraints and vice versa. We tried to keep semantics of ORM constraints within SCM, but it turned out that some of them are hard to use in textual notation in initial form. For instance, SCM functional constraint template is more suitable for textual notation than ORM uniqueness constraint template because the former has well known textual representation as “ $x \rightarrow y$ ”, while the latter has no it.

Also we aimed at reducing quantity of self-meaning modeling elements and at generalizing complex and simple constraints to one notation. For instance, SCM inheritance is an association with equivalence and mandatory constraints, there is no inheritance as a self-meaning element. Another example is the constraint “[(Person–Solution Act–Task): Solution Act \equiv (Person, Task)]”. Here existing equivalence template is used instead of introducing a special uniqueness template spanning different associations (which exists in ORM). Moreover, the notation is easily generalized to complex constraints by using a complex query instead of a simple one.

Simple constraints are not sufficient for formalization of all business rules existing in application domains. For instance, the constraints “Percent complete is a ratio between total time amount of solution acts made for this assignment and its own time amount” and “Assignment cannot be reassigned to another person or task” cannot be formulated as simple ones.

3. SCQL expression structure

SCQL [Ov05a, Ov05b] queries are core of SCM constraints. A SCQL expression is a tree of the operations: composition, transformation, union, minus, logical selection, association selection. The first four are non-leaf operations, and the last two are leaves.

Main distinctive property of SCQL is that it does not use proper association names for query formulation. Formulating a query, one operates application domain concepts and facts of their interconnection purely. For instance, to query an association, one enumerates its concepts: (Person, Assignment). Another distinctive property is that SCQL queries have unordered semantic signatures (in other words, signatures are sets of domain concepts). Both properties make the language simpler and more transparent.

SCQL composition is the only way to join subqueries and can be thought as a mathematical superposition or a natural join. Therefore, SCQL does not use explicit join predicates at all. Composition has several notations, two of which are used in the running example. General notation is enumeration of subqueries being composed using comma (see lines 12-14 in the example). And path expression notation is used for associations only and represents chaining concepts using dash: (Person–Solution Act–Task). This query is equivalent to ((Person, Solution Act), (Solution Act, Task)).

Transformation is the only way to change signature of a subquery controllably. Transformation defines a result signature explicitly, making projection, calculation of aggregate and non-aggregate functions. In case of aggregate function calculation, group-by clause is never used, as grouping is always done by all not aggregated result concepts. For instance, the query SELECT Assignment, SUM(Time Amount) FROM (Assignment–Solution Act–Time Amount) calculates total time amount consumed for solving each assignment. The same query can be written down using dot notation as follows: (Assignment–Solution Act–Time Amount). (Assignment, SUM(Time Amount)). Both notations are absolutely interchangeable.

Union and minus also have specificity: signatures of subqueries can be absolutely different. A logical selection represents a concept-based predicate used as subquery of a composition; as a result, the composition is additionally filtered according to all its nested logical selections. Composition with nested logical selections is the only way to filter subqueries in SCQL (selection as such does not exist).

Sometimes one concept is used within one query several times in different roles, and different roles of the concept are considered to be different query columns. Role concepts are used in such cases. A role concept represents a domain concept extended with a role name in round brackets after it. For instance, the constraint at lines 12-15 of the running example uses the concept Time Amount in two roles: “Time Amount(Assignment)” and “SUM(Time Amount)” where the second role concept is the result of aggregate function calculation.

SCQL has additional context (closure) mechanism which permits using several indirectly associated concepts as if they are associated directly [Ov04, Ov05a], but the mechanism is out of the scope of the paper. We believe all the properties make possible using SCQL by non-technical persons. So, SCQL well suits the purpose of constraint formulation as SCM model becomes relatively simple to read and write.

4. Complex and Dynamic SCM Constraints

In the most general case any constraint represents a predicate defined over queries of two types: a) executed within the current model state, b) executed within the next model state after the current transaction commits. A constraint is static if it is based on current state only, otherwise it is dynamic. For instance, the lines 12-17 of the running example define a static constraint, and the lines 18-20 define dynamic one.

Dynamic constraints are not reflected within present conceptual modeling techniques (ORM, ER, FCO-IM, and others). Dynamic constraints mean restriction of possible changes of a conceptual model's content. For instance, the constraint 18-20 forbids changing person and task of existing assignments. These constraints are very fine business rules which often appear formally only at implementation phase. SCM takes attempt to define them at the conceptual design stage. Within a SCM constraint, if a query is to be done to a next model state, the sign “^” is placed before the query. So, the query (^(*Person-Assignment-Task*), (*Assignment*)) is composition of the subquery ^(*Person-Assignment-Task*), selection of all assignments with their persons and tasks from the next model state, and the subquery (*Assignment*), selection of all existing assignments from the current model state. The result of the composition is all assignments with their persons and tasks (according to the next state) which exist in the current and next states.

The most general way of SCM constraint formulation is use of SCQL-extended predicate calculus which considers concepts as sets and SCQL queries as relations. For instance, the static template-based constraint [(*Person-Assignment-Task*): *Assignment* ≡ (*Person*, *Task*)] can be reformulated using general predicate notation as

$$\left[\begin{array}{l} \forall (Person - Assignment - Task) \\ \left\{ \begin{array}{l} \exists_1 Assignment_2 \{ \exists (Person - Assignment_2 - Task) \} \wedge \\ \exists_1 (Person_2, Task_2) \{ \exists (Person_2 - Assignment - Task_2) \} \end{array} \right\} \end{array} \right]$$

Predicate notation is used when there is no another more compact way to formulate a constraint. Detailed description of the calculus is out of the scope of the paper.

The special case of static constraints is instance-level constraints which are predicates based on a query result' instances. Such constraints have a special syntax described in the section 2 and imply for-all-instances clause in the very beginning. For example, the instance-level static constraint [(*Person-Assignment-Task*): *Assignment* ≡ (*Person*, *Task*)] reads as “for all instances of the query (*Person-Assignment-Task*) it is true that *Assignment* ≡ (*Person*, *Task*)”.

Complex constraint is a constraint which is based on at least one complex query (see the section 2 for its definition) or uses predicate notation. Such constraints are called complex as it is difficult to represent them compactly in graphical notation. As dynamic constraints, complex constraints are not covered by existing conceptual modeling techniques. According to the foresaid, all constraints discussed at the section 2 are simple static.

All constraint templates introduced at the section 2 and many others can be used for complex and dynamic constraint formulation using the same notations as simple constraints. For instance, consider in details a complex constraint at lines 12-15 of the running example

```

[(Percent Complete–Assignment–Time Amount(Assignment)),
(SELECT Assignment, SUM(Time Amount)
FROM (Assignment–Solution Act–Time Amount)):
Percent Complete=Time Amount(Assignment)/SUM(Time Amount)]

```

which means “percent complete is a ratio between total time amount of solution acts made for this assignment and its own time amount”. The constraint is executed in the following stages:

- a) (Percent Complete–Assignment–Time Amount(Assignment)) is the composition of two associations (Percent Complete, Assignment) and (Assignment, Time Amount), then the concept “Time Amount” is extended to the role concept “Time Amount(Assignment)”.
- b) (SELECT Assignment, SUM(Time Amount) FROM (Assignment–Solution Act–Time Amount)) is initial composition of two associations (Assignment, Solution Act) and (Solution Act, Time Amount), then projection on “Assignment” and “Time Amount”, then grouping on “Assignment”, and then calculation sum of all time amounts within each group.
- c) Then the subqueries (a) and (b) are composed, considering “Time Amount(Assignment)” and “SUM(Time Amount)” as different role concepts.
- d) Then for each instance of (c) it is asserted that “Percent Complete=Time Amount(Assignment)/SUM(Time Amount)”.

Now consider in details the dynamic constraint at lines 18-20: [(^(Person–Assignment–Task), (Assignment)) is part of (Person–Assignment–Task)]. The constraint means “assignment cannot be reassigned to another person or task” and has the following execution stages:

- a) ^(Person–Assignment–Task) is the composition of two associations (Person, Assignment) and (Assignment, Task) both taken from the next model state.
- b) (Assignment) is selection of all assignments existing in the current model state.
- c) Then (a) and (b) are composed, considering both occurrences of “Assignment” as the same role concept with empty name.
- d) (Person–Assignment–Task) is the same composition as (a), but both associations are taken from the current model state.
- e) Then it is asserted that each instance of (c) exists among instances of (d).

5. Conclusion

SCM takes attempt to fill up the gap of present conceptual modeling approaches which have no support for complex and dynamic constraints now. Constraint formulation within SCM inherits all useful properties of SCQL main of which are that it does not use proper association names for query formulation and has simple and transparent structure. Main property of SCM itself significantly affects the process of constraint formulation as well. The property is association completeness which means that each association completely defines interconnection of underlying concepts; one can think about application domain in the plane of concepts purely. As a result, we believe that non-technical and technical persons can be relatively easily studied to SCQL and SCM, including complex and dynamic constraint formulation.

SCM modeling approach is evolving. But we try to hold the technique at the most abstract level, not adding features concerning implementation even slightly. Each extension is checked for necessity and is done only if the necessity is proven.

SCM and SCQL are used as foundation for SCM-based client/server technology [Ov05b]. Further development of SCM in this direction will result in a system integrating several sources of structural information under one SCM-based data access interface.

Another direction of SCM development is creation of conceptual modeling tools providing the most abstract description of application domains and subsequent mapping to models/query languages/programming languages reflecting more implementation details. At this role, we suppose SCM can be used as a top level Computational Independent Model of OMG Model-Driven Architecture [OMG03a].

References

- [BH97] Bloesch A.C., Halpin T.A. Conceptual Queries using ConQuer-II // Proceedings of ER'97: 16-th International Conference on Conceptual Modeling, 1997, pp. 113-126.
- [Ch76] Chen P.P.S. The entity-relationship model – towards a unified view of data // ACM Transactions on Database Systems, 1(1), 1976, pp. 9-36.
- [Ha97] Halpin T.A. Modeling for Data and Business Rules (interview ed. R.G. Ross) // Data Base Newsletter, 25(5), 1997.
- [Ha01] Halpin T.A. Information Modeling and Relational Databases. Morgan Kaufmann, San Francisco, 2001.
- [Ha05] Halpin T.A. Objectification // In (Castro J., Teniente E. eds) Proc. of CAiSE'05 Workshops Vol. 1, Porto, FEUP, 2005, pp. 617-628.
- [HPW93] ter Hofstede A.H.M., Proper H.A., van der Weide. Formal Definition of a Conceptual Language for the Description and Manipulation of Information Models // Information Systems, 18(7), 1993, pp. 489-523.
- [HPW96] ter Hofstede A.H.M., Proper H.A., van der Weide. Query Formulation as an Information Retrieval Problem // The Computer Journal, 39, 1996, pp. 255-274.
- [OMG03a] OMG Model Driven Architecture Guide V1.0.1 (<http://www.omg.org/cgi-bin/doc?omg/03-06-01>), 2003.
- [OMG03b] OMG UML 2.0 Infrastructure Specification (<http://www.omg.org/cgi-bin/doc?ptc/2003-09-15>), 2003.
- [OMG03c] OMG UML 2.0 OCL Specification (<http://www.omg.org/cgi-bin/doc?ptc/2003-10-14>), 2003.
- [OMG04] OMG UML 2.0 Superstructure Specification (<http://www.omg.org/cgi-bin/doc?ptc/2004-10-02>), 2004.
- [Ov04] Ovchinnikov V.V. A Conceptual Modeling Technique Based on Semantically Complete Model, its Applications // In (Doroshenko A., Halpin T., Liddle S., Mayr H. eds.) Proc. of the 3rd International Conference ISTA'2004: Information Systems Technology and its Applications, Salt Lake City, GI Lecture Notes in Informatics P-48, 2004, pp. 25-38.
- [Ov05a] Ovchinnikov V.V. A Concept-Based Query Language Not Using Proper Relation Names // In (Castro J., Teniente E. eds) Proc. of CAiSE'05 Workshops Vol. 1, Porto, FEUP, 2005, pp. 617-628.
- [Ov05b] Ovchinnikov V.V. SCM Portal (<http://scm.lipetsk.ru>), 2005.