Static dependency analysis for semantic data validation

D.V. Ilyin <denis.ilyin@ispras.ru>
N.Yu. Fokina <nfokina@ispras.ru>
V.A. Semenov <sem@ispras.ru>
Ivannikov Institute for Systems Programming of the RAS,
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia

Abstract. Modern information systems manipulate data models containing millions of items, and the tendency is to make these models even more complex. One of the most crucial aspects of modern concurrent engineering environments is their reliability. The principles of ACID (atomicity, consistency, isolation, durability) are aimed at providing it, but directly following them leads to serious performance drawbacks on large-scale models, since it is necessary to control the correctness of every performed transaction. In the paper, a method for incremental validation of object-oriented data is presented. Assuming that a submitted transaction is applied to originally consistent data, it is guaranteed that the final data representation is also consistent if only the spot rules are satisfied. To identify data items subject to spot rule validation, a bipartite data-rule dependency graph is formed. To automatically build the dependency graph a static analysis of the model specifications is proposed to apply. In the case of complex object-oriented models defining hundreds and thousands of data types and semantic rules, the static analysis seems to be the only way to realize the incremental validation and to make possible to manage the data in accordance with the ACID principles.

Keywords: information systems; ACID; data consistency management; EXPRESS

DOI: 10.15514/ISPRAS-2018-30(3)-19

For citation: Ilyin D.V., Fokina N.Yu., Semenov V.A. Static dependency analysis for semantic data validation. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 271-284. DOI: 10.15514/ISPRAS-2018-30(3)-19

1. Introduction

Management of semantically complex data is one of the challenging problems tightly connected with emerging information systems such as concurrent engineering environments and product data management systems [1-4]. Although transactional guarantees ACID (Atomicity, Consistency, Isolation, and Durability) are widely recognized and recommended for any information system, it is difficult to maintain the consistency and integrity of data driven by complex object-oriented models. Often such models are specified in EXPRESS language being part of the STEP standard on

industrial automation systems and integration (ISO 10303). To be unambiguously interpretable by different systems the data must satisfy numerous semantic rules imposed by formal models. Maintaining data consistency and ensuring system interoperability become a serious computational problem. Full semantic validation requires extremely high costs, often exceeding the processing time of individual transactions. Periodic validation is possible, but at a high risk of violating rules and losing actual data.

The paper presents an effective method for incremental validation of object-oriented data. An idea of incremental checks is well-understood and was successfully implemented for the validation of such specific data as UML charts, XML documents, deductive databases [5-7]. Unlike the aforementioned results, the proposed method can be applied to semantically complex data driven by arbitrary object-oriented models.

Assuming that a submitted transaction is applied to originally consistent data, it is guaranteed that the final data representation is also consistent if only the spot rules are satisfied. To identify data items subject to spot rule validation, a bipartite data-rule dependency graph is formed. To automatically build the dependency graph a static analysis of the model specifications is proposed to apply. In the case of large-scale models defining hundreds and thousands of data types and semantic rules, static analysis seems to be the only way to realize the incremental validation and to make possible to effectively manage the data in accordance with the ACID principles.

The structure of the paper is as follows. In section 2, we will shortly overview EXPRESS language with an emphasis on the data types and the rule categories admitted by the language. Formal definitions of model-driven data, rules and transactions are also provided. In section 3, we will present a complete validation routine and then explain how an incremental validation can be arranged using the proposed dependency graph. This is accompanied by an example of the model specification. In conclusion, we summarise benefits of the proposed validation method and outdraw future efforts.

2. Product data and transactions

2.1 EXPRESS language

Product data models and, particularly, semantic rules can be specified formally in EXPRESS (ISO 2004) language [8]. This object-oriented modeling language provides a wide range of declarative and imperative constructs to define both data types and constraints imposed upon them. The supported data types can be subdivided into the following groups: simple types (character, string, integer, float, double, Boolean, logical, binary), aggregate types (set, multi-set, sequence, array), selects, enumerations, and entity types.

Depending on the definition context, three basic sorts of constraints are distinguished in the modeling language: rules for simple user-defined data types, local rules for

object types, and global rules for object type extents. Depending on the evaluation context these imply the following semantic checks:

- attribute type compliance (R_0);
- limited widths of strings and binaries (R_1, R_2);
- size of aggregates (R_3);
- multiplicity of direct and inverse associations in objects (R_4, R_5);
- uniqueness of elements in sets, unique lists and arrays (R_6);
- mandatory attributes in objects (R_7);
- mandatory elements in aggregates excluding sparse arrays (R_8);
- value domains for primitive data types (R_9);
- value domains restricting and interrelating the states of separate attributes within objects (R_{10} or so-called local rules);
- uniqueness of attribute values (optionally, their groups) on object type extents (R_{11} or uniqueness rules);
- value domains restricting and interrelating the states of whole object populations (R_{12} or so-called global rules). Value domains can be specified in a general algebraic form by means of all the variety of imperative constructs available in the language (control statements, functions, procedures, etc.).

Certainly, each product model defines own data types and rules. Therefore, semantic validation methods and tools should be developed in a model-driven paradigm allowing their application for any data whose model is formally specified in EXPRESS language. For a more detailed description refer to the mentioned above standard family which regulates the language.

2.2 Formalization of models, data and transactions

An object-oriented data model M can be formally considered as a triple $M = \langle T \cup \prec \cup R \rangle$, where the types $T = \{C \cup S \cup A \cup \dots\}$ are classes C, simple types S, aggregates A and other constructed structures allowed by EXPRESS. Generalization/specialization relations $\prec$ are defined among these types. Each class $c \in C$ defines a set of attributes in the form $c.a: C \mapsto T$. The attributes $c.a: C \mapsto C$, $c.a: C \mapsto aggregate(C)$ are single and multiple associations which play role of object references. The rules $R = \{R_0 \cup R_1 \cup R_2 \cup \dots \cup R_{12}\}$ define the value domains of typed data in an algebraic way in accordance with EXPRESS. The rules are subdivided into 12 categories enumerated above. Let us define the key concepts that are used in further consideration.

An object-oriented dataset $x = \{o_1, o_2, \dots\}$ is said to be driven by the model $M\langle T, \prec, R\rangle$ if all the objects belong to its classes: $\forall\, o \in x \to typeof(o) \in C \subset T$.

Let a dataset x is driven by the model $M\langle T, \prec, R\rangle$. All the objects $\{o^*\} \subset x$ such that $subtypeof(o^*) = c \in C \subset T$ are called an extent of the class c on the dataset x. A query returning the class extent c on the dataset x is called the extent query and is designated as $Q_{extent}(x, c)$.

Let a dataset x is driven by the model $M\langle T, \prec, R\rangle$. An object set $\{o^*\} \subset x$, $typeof(o^*) = c^* \in C \subset T$ is said to be interlinked with the objects $\{o\} \subset x$, $typeof(o) = c \in C \subset T$ along the association $c.a$ if $\forall o \in \{o\}, o.a \subset \{o^*\}$, $\forall o^* \in \{o^*\} \to \exists o \in \{o\}: o^* \in o.a$. We will denote that as $\{o\} \xrightarrow{c.a} \{o^*\}$.

Let a dataset x is driven by the model $M\langle T, \prec, R\rangle$. An object set $\{o^*\} \subset x$, $typeof(o^*) = c^* \in C \subset T$ is said to be interlinked with the objects $\{o\} \subset x$, $typeof(o) = c \in C \subset T$ along the route $\{c.a\}$ if $\exists\, \{o'\} \subset x, \{o''\} \subset x$, …, so that $\{o\} \xrightarrow{c.a} \{o'\} \xrightarrow{c'.a'} \{o''\} \xrightarrow{c''.a''} \dots \to \{o^*\}$. A query returning the objects $\{o^*\}$ interlinked with a given set $\{o\}$ along the route $\{c.a\}$ is called the route query and is designated as $Q_{route}(x, \{o\}, \{c.a\})$. A query returning the objects $\{o\}$ by a given object set $\{o^*\}$ is called the reverse route query and is designated as $Q_{route}(x, \{o^*\}, rev\,\{c.a\})$.

The object set $x = \{o_1, o_2, \dots\}$ driven by the model $M\langle T, \prec, R\rangle$ is called consistent if all the rules being instantiated and evaluated are satisfied on this data set: $\forall\, r \in R \to r(x) = true$.

Finally, let us introduce the concept of the delta as a specific representation of transactions. Each delta $\Delta(x', x)$ aggregates the changes happened in the dataset $x' = \{o'_1, o'_2, \dots\}$ compared with its original revision $x = \{o_1, o_2, \dots\}$. It is assumed that both revisions are driven by the same model and the objects have unique identifiers that allows to uniquely map the objects and to compute delta in a formal way as $\Delta(x', x) = delta(x', x)$. The delta can be arranged as bidirectional one and then any of the revisions can be restored by the given other: $x' = apply(x, \Delta)$ and $x = apply(x', \Delta^{-1})$.

The delta is represented as a set of elementary and compound changes $\Delta = \{\delta\}$, where each change can be either the creation of an object, or its deletion or modification designated as $\delta_{new(o)}$, $\delta_{del(o)}$, $\delta_{mod(o)}$ correspondingly. The modification, in turn, is represented as a change in the attributes $\delta_{mod(o)} = \{\delta_{mod(o.a)}\}$ that in the case of aggregates is represented by the operations of insertion, removal and modification of the elements $\delta_{mod(o.a)} = \{\delta_{ins(o.a[])}, \delta_{rem(o.a[])}, \delta_{mod(o.a[])}\}$. In what follows, we assume that each creation operation in the delta representation is complemented by the operations of initializing the attributes that are equivalent to the modification operations. Each deletion operation is supplemented by the operations of resetting the attributes to an undefined state, also representable by the modification operations. Regardless of the way, the delta is structured, only elementary operations are taken into account in the context of the studied validation problems.

3. Validation

3.1 Complete validation

The complete validation routine is provided below (see Figure 1). In a cycle on all objects their attributes are checked against the rules of the categories $R_1 \cup R_2 \cup \ldots \cup R_9$. The checks are performed individually for each attribute provided that the corresponding rules are imposed on their types. In case of detected violations, the error messages are logged. Rules R_{10} are evaluated for entire objects in the same loop. The second cycle is formed due to the need for checks of uniqueness rules R_{11}. Since these rules are declared inside the class definitions, an additional cycle is arranged on the model classes. The rules are evaluated on the class extents. Finally, the third cycle allows to check global rules R_{12} which are defined directly in the model. Such checks are performed for the corresponding class extents.

```
for each object o ∈ x in dataset
    for each attribute o.a in object
        for each attribute rule ∈R0 U R1 U R2 U … U R9 defined for typeof( o.a )
            check rule(o.a), log if violated
    for each local rule ∈ R10 defined for typeof( o )
        check rule( o ), log if violated
for each class c ∈ C defined in model
    for each uniqueness rule ∈ R11 defined for class c
        check rule( Q_extent( x, rule.c ) ), log if violated
for each global rule ∈ R12 defined in model
    check rule( Q_extent( x, rule.c1 ), Q_extent( x, rule.c2 ),… ), log if violated
```

Fig. 1. Complete validation routine

As mentioned above, complete validation of semantically complex product data is a computationally costly task that can cause performance degradation when processing transactions. Incremental validation makes it possible to reduce the amount of checks to be performed.

3.2 Incremental validation

The proposed incremental validation method is based on the idea of localizing spot rules that can be affected by a transaction and generating a set of semantic checks that is sufficient to detect all potential violations. For this purpose, the dependency graph is built by a given specification of the data model in EXPRESS language. For brevity, we just explain that this structure represents and omit the details of how it can be formed using static analysis of the specification.

The dependency graph is a bipartite graph whose nodes represent the kinds of transaction operations and the categories of semantic rules both defined by the underlying model. An operation node is connected with the rule nodes by directed edges if only such operations can violate the rules being instantiated for particular data. Usually, the semantics of the operations imply what are the data it is applied to. Sometimes the inspected data are apriori unknown and have to be determined by

executing corresponding route queries. Therefore, each edge is formed by the dependency structure σ containing both a rule reference $\sigma.rule$ and an optional query route $\sigma.route$. In some sense, the graph reflects the transaction structure as if it contains all possible kinds of changes and the data organisation as if all data types are present and all rules are potentially suffered to violations. As mentioned above, only elementary operations are involved in the dependency analysis.

Thus, the dependency graph enables to determine spot rules that could be violated for particular data due to the accepted transaction. For example, if the node operation is a modification of the object attribute $c.a$ and a rule $r \in R_0 \cup R_1 \cup R_2 \cup \ldots \cup R_9$ is defined for its type, then the node $\delta_{mod(c.a)}$ is connected with the rule node r by a corresponding edge. Having a specific operation of this kind $\delta_{mod(o.a)}$, $typeof(o) = c$ in the delta representation the corresponding check $r(o.a)$ can be produced using the dependency edge.

The method of the dependency graph construction is described in more detail in the next section. Still, here we will point out some of its important features.

If the same attribute $c.a$ participates in an expression of the domain rule $r \in R_{10}$ for the class c, then the operation $\delta_{mod(o.a)}$, $typeof(o) = c$ produces the check $r(o)$ for the object o.

If the attribute $c.a$ participates in the uniqueness rule $r \in R_{11}$ defined for the class c, then another dependency edge must be associated with the operation node. In this case, the corresponding check $r(Q_{extent}(x, c))$ must be performed.

There is a more difficult case when the attribute $c.a$ participates in an expression of the domain rule $r \in R_{10}$ defined for the other class c^*. The attribute $c.a$ is assumed to be accessed by traversing associated objects along the route $\{c^*.a^*\}$ from the objects $o^* \in c^*$. Then the operation $\delta_{mod(o.a)}$, $typeof(o) = c$ induces the checks $r(o^*)$ for all $o^* \in Q_{route}(x, o, rev\ \{c^*.a^*\})$. To identify and perform such checks the operation node must be connected with the evaluated rule node and a route $\{c^*.a^*\}$ must be prescribed to the edge. The dependency analysis of spot rules $r \in R_{12}$ is carried out in a similar way.

Finally, we note that the operations of creating and deleting objects on the assumptions made above can only violate global rules and only in those cases if the cardinalities of class extents are computed. Considering object references as specific attribute types, it is possible to localize some spot rules more exactly. Differing operations on aggregates also leads to better localization of spot rules. For brevity we omit the details how the spot rules can be localized more carefully and provide an example in the next subsection.

```
for each elementary operation δ(o),δ(o.a) ∈ delta
    { σ } = dependency_graph( kindof( δ ) )
    for each dependency σ ∈ { σ }
        switch kindof( σ.rule )
            case attribute_rule :
                check σ.rule( o.a ), log if violated
            case local_rule :
```

```
                { o* } = Query_route( x, o, rev (σ.route) )
                for each o* ∈ { o* }
                    checkset.put( σ.rule( o* ) )
            case uniqueness_rule :
                checkset.put( σ )
            case global_rule :
                checkset.put( σ )
    for each check σ, σ(o) ∈ checkset
        switch kindof( σ.rule )
            case local_rule :
                check σ.rule( o ), log if violated
            case uniqueness_rule :
                check σ.rule( Query_extent( x, σ.rule.c ) ), log if violated
            case global_rule :
        check σ.rule( Query_extent( x, σ.rule.c1 ), Query_extent( x, σ.rule.c2 ),… ),
    log if violated
```

Fig. 2. Incremental validation routine

The validation routine presented in Figure 2 consists in the sequential traversing of delta operations, determining the nodes of the operation semantics, obtaining associated spot rule nodes, evaluating the rules directly or filling the checkset for the subsequent validation. The checkset is organized as an indexed set of records each of which stores references on the validated rule, query and factual data to perform the corresponding check. The use of the checkset is motivated by the fact that some operations lead to repeated checks of the same rules. Indexing of the checkset allows you to exclude repeated records and, thus, to avoid redundant computations. At the same time, the attribute rule checks are always produced once by the modification operations and, therefore, it is more expedient to execute them immediately, without overloading the checkset.

3.3 Dependency graph construction

To construct the dependency graph, an abstract syntactic tree for the model is built. According to the retrieved data, for all attribute declarations operation nodes are built. Number and types of these nodes constructed for a single attribute depend on its type. For non-aggregate attributes $c.a$ only node $\delta_{mod}(c.a)$, representing modification of the attribute, is built. For aggregate attributes $c.a[]$ three nodes are created: (1) $\delta_{ins}(c.a[])$ – insertion of a new element; (2) $\delta_{mod}(c.a[])$ – modification of an element of the aggregate; (3) $\delta_{rem}(c.a[])$ – removal of an element.

Construction of the dependency graph proceeds with generating rule nodes. We handle construction of nodes for rules R_1-R_9 and R_{10}-R_{12} differently.

For rules R_1-R_9 we take all explicit attributes and build rule nodes for each of them. The types of rule nodes depend on the type of the attribute in question. For instance, if it is a bounded string c.*S*, we generate a $R_1(c.S)$ (R_1 – limited width of strings), connected with the node corresponding to the modification of S $\delta_{mod}(c.S)$. Similarly, if an attribute is a bounded aggregate, we construct a node of type R_4 and connect it with the insertion $\delta_{ins}(c.a[])$ and/or removal $\delta_{rem}(c.a[])$ operation nodes of the

attribute, depending on the side from which the aggregate is bounded – if it is bounded above, then only with insertion node, if below – with removal, if from both sides – with both of them.

The way of construction of rule nodes for R_{10}-R_{12} is uniform. We start with locating all local rules for R_{10}, all uniqueness rules for R_{11} and all global rules for R_{12}. For each of the rules, we find all attributes used in it. If an attribute is explicit, we only connect its modification with the rule node, and also with insertion and removal, if it is an aggregate used inside a SIZEOF operation. If an attribute is derived, we take its definition and find the attributes used in it; if inverse – we proceed with analyzing the attribute it references. For derived and explicit attributes, the analysis is performed recursively, until all the explicit attributes, directly and indirectly referenced by them, are located. Then all of them are connected with the rule node corresponding to the rule in question. If the during the analysis we find a node that is a function call, we substitute its formal parameters with actual and thus locate the attributes which are used in it; the analysis of a function body with the parameters substituted is completely identical to the analysis of a rule.

An example illustrating the constructed graph is given in the next subsection.

3.4 Example of a dependency graph

Let us consider a fragment of the EXPRESS specification of a project management system. Three classes depicted in Figure 3 – *Task*, *Link* and *Calendar* – are its core entities. The meaning of Task is self-evident; *Link* represents a connection defining a relation and execution order between two tasks. The fact that between two tasks might be only a single link of one type is reflected in uniqueness rule *ur1*. A *Calendar* defines a typical working pattern: working days, working times, holidays. The calendar can be assigned to specific tasks, and one calendar can be set as a default project calendar, that means that it will be used for tasks for which no task calendar is set. Besides that, it is possible to use an *Elapsed* calendar for a task implying that work will be performed 24/7. Global rule *SingleProjectCalendar* restricts the possible number of project calendars to no more than one. Moreover, local rule *wr3* is used to check that if a task has got a task calendar, it the reference to it must be non-null. One more local rule, *wr2*, restricts the length of an *EntityName* to be between 1 and 32 characters.

```
TYPE LinkEnum = ENUMERATION OF
    (START_START, START_FINISH, FINISH_START, FINISH_FINISH);
END_TYPE;

TYPE CalendarRuleEnum = ENUMERATION OF
    (TASK, PROJECT, ELAPSED);
END_TYPE;

FUNCTION TaskIsCyclic (T1 : Task, T2 : Task) : BOOLEAN;
    IF (SIZEOF(T1.Parent) = 0) THEN RETURN(FALSE);
    ELSE
        IF ((TaskIsCyclic(T1.Parent[1], T2) = TRUE) OR (T1 = T2))
```

```
            THEN RETURN(TRUE);
        END_IF;
    END_IF;
END_FUNCTION;
RULE SingleProjectCalendar FOR (Calendar);
WHERE
    wr1: SIZEOF(QUERY(Temp <* Calendar | Temp.isProjectCalendar =
TRUE)) <= 1;
END_RULE;

TYPE EntityName = STRING;
WHERE
    wr2: (1 <= SELF) AND (SELF <= 32);
END_TYPE;

ENTITY Task;
    ID : INTEGER;
    Name : EntityName;
    TaskCalendar : Calendar;
    CalendarRule : CalendarRuleEnum;
    Children : LIST [0:?] OF Task;
DERIVE
    TaskDuration : Duration := ?;
INVERSE
    Parent : SET [0:1] OF Task FOR Children;
    DownstreamLinks : SET [0:?] OF Link FOR Predecessor;
    UpstreamLinks : SET [0:?] OF Link FOR Successor;
WHERE
    wr3 : CalendarRule <> CalendarRuleEnum.TASK OR
EXISTS(TaskCalendar);
    wr4 : (SIZEOF(Parent) = 0)  OR (TaskIsCyclic(Parent[1], SELF) =
FALSE);
UNIQUE
    ur1 : ID;
END_ENTITY;

ENTITY Link;
    ID : INTEGER;
    LinkType : LinkEnum;
    Predecessor : Task;
    Successor : Task;
UNIQUE
    ur2 : LinkType AND Predecessor.ID AND Successor.ID;
    ur3 : ID;
END_ENTITY;

ENTITY Calendar;
    ID : INTEGER;
    Name : OPTIONAL EntityName;
    IsProjectCalendar : BOOLEAN;
UNIQUE
    ur4 : ID;
END_ENTITY;
```

Fig. 3. An example of the model specification in EXPRESS language

The dependency graph for this fragment of the specification is shown in Figure 4.

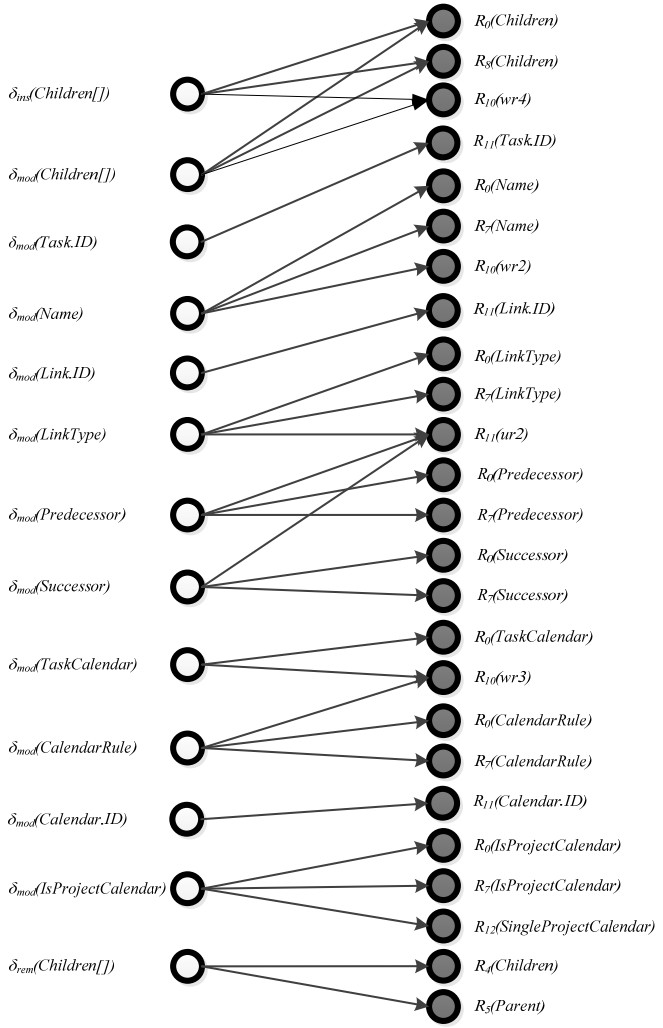

Fig. 4. A fragment of the model dependency graph

Each operation of attribute modification except for removal of elements from the list of task children is connected with the rules validating corresponding attribute type compliance R_0 and availability of defined values for mandatory attributes R_7. To avoid placement of null values to the list of mandatory elements the rule R_8 should be validated as well after the operations have been performed. The insertion cannot violate multiplicity of the direct and inverse associations as their upper borders are unlimited, but checks R_4, R_5 should be performed when an element is removed from

Children. Therefore, the corresponding operation nodes should be connected with the aforementioned nodes of the rules that the operations may potentially violate. As the expression for the local rule *wr3* includes the attributes *CalendarRule* and *TaskCalendar*, the nodes corresponding to the operations of modification of these attributes are connected with the *wr3* rule node. For the rule *wr2* defining the value range of the *EntityName* type, there is a connection between the *EntityName* modification node and the *wr2* rule node. The corresponding edges are assigned by the routes by traversing of which the attributes could be accessed. The expression of the global rule *SingleProjectCalendar* references only one attribute *IsProjectCalendar*, so the appropriate graph nodes are connected by the edge as well. Modification of any attribute of the *Link* class can affect its uniqueness defined by *ur2*; hence the connections between *LinkType*, *Predecessor* and *Successor* and the uniqueness rule node.

It is also possible that a change affects a constraint not directly but through an inverse association, or even a chain of them, where other classes can be involved. In this case, rules for all the chain of affected classes is added to the checkset. Furthermore, they can be affected not only by direct associations but also by the inverse. For instance, cardinality constraints on inverse aggregate attributes causes insertion of additional rule nodes to the graph.

4. Conclusion

This paper presents the incremental method of model data validation. The method is applicable for semantically complex data driven by arbitrary object-oriented models. It allows to increase the performance of semantic validation and to effectively manage the data in accordance with the ACID principles.

The planned work concerns basically the implementation of the method proposed and its evaluation for industry meaningful product data. The expected positive results will allow its wide introduction into new software engineering technologies and emerging information systems.

References

[1]. V.A. Semenov. Product Data Management with Solid Transactional Guarantees, In Transdisciplinary Engineering: A Paradigm Shift Series Advances in Transdisciplinary Engineering, IOS Press, 2017, pp. 592-599.

[2]. L. Lämmer and M. Theiss. Product Lifecycle Management, In Concurrent Engineering in the 21st Century – Foundations, Developments and Challenges, Springer, 2015, pp. 455-490.

[3]. J. Osborn. Survey of concurrent engineering environments and the application of best practices towards the development of a multiple industry, multiple domain environment. Clemson University, 2009. Accessed: 29/01/2018. Available: http://tigerprints.clemson.edu/all_theses/635/

[4]. M. Philpotts. An introduction to the concepts, benefits and terminology of product data management, Industrial Management & Data Systems, MCB University Press, vol. 96, no. 4, 1996, pp. 11–17.

[5]. X. Blanc, A. Mougenot, I. Mounier, T. Mens. Incremental Detection of Model Inconsistencies based on Model Operations. In Advanced Information Systems Engineering, CAiSE 2009, LNCS, vol. 5565, Springer, 2009, pp. 32-46.

[6]. C. Xu, C.S. Cheung, W.K. Chan. Incremental Consistency Checking for Pervasive Context. In Proc. the 28th International Conference on Software Engineering, 2006, pp. 292-301.

[7]. J. Harrison, S.W. Dietrich. Towards an Incremental Condition Evaluation Strategy for Active Deductive Databases. In Research and Practical Issues in Databases, World Scientific, 1992, pp. 81-95.

[8]. ISO 10303-11: 2004. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual, ISO, 2004.

Статический анализ зависимостей для семантической валидации данных

Ильин Д.В. <denis.ilyin@ispras.ru>
Фокина Н.Ю. <nfokina@ispras.ru>
Семенов В.А. <sem@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Современные информационные системы манипулируют моделями данных, содержащими миллионы объектов, и тенденция такова, что эти модели постоянно усложняются. Одним из важнейших аспектов современных параллельных инженерных сред является их надежность. Принципы ACID (атомарность, согласованность, изолированность, устойчивость) направлены на ее обеспечение, однако прямое следование им приводит к серьезному снижению производительности на крупномасштабных моделях, поскольку необходимо контролировать правильность каждой выполненной транзакции. В настоящей статье представлен метод инкрементальной валидации объектно-ориентированных данных. Предполагая, что транзакция применяется к первоначально согласованным данным, гарантируется, что окончательное представление данных также будет согласованным, если только будут выполнены локальные правила. Для определения объектов данных, подлежащих проверке, формируется двудольный граф зависимостей по данным. Для автоматического построения графа зависимостей предлагается применять статический анализ спецификаций модели. В случае сложных объектно-ориентированных моделей, включающих сотни и тысячи типов данных и семантических правил, статический анализ, по-видимому, является единственным способом реализации инкрементальной валидации и обеспечения возможности управления данными в соответствии с принципами ACID.

Ключевые слова: информационные системы; ACID; управление целостностью данных; EXPRESS

DOI: 10.15514/ISPRAS-2018-30(3)-19

Для цитирования: Ильин Д.В., Фокина Н.Ю., Семенов В.А. Статический анализ зависимостей для семантической валидации данных. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 271-284 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-19

Список литературы

[1]. V.A. Semenov. Product Data Management with Solid Transactional Guarantees, In Transdisciplinary Engineering: A Paradigm Shift Series Advances in Transdisciplinary Engineering, IOS Press, 2017, pp. 592-599.

[2]. L. Lämmer and M. Theiss. Product Lifecycle Management, In Concurrent Engineering in the 21st Century – Foundations, Developments and Challenges, Springer, 2015, pp. 455-490.

[3]. J. Osborn. Survey of concurrent engineering environments and the application of best practices towards the development of a multiple industry, multiple domain environment. Clemson University, 2009. Дата обращения: 29/01/2018. Режим доступа: http://tigerprints.clemson.edu/all_theses/635/

[4]. M. Philpotts. An introduction to the concepts, benefits and terminology of product data management, Industrial Management & Data Systems, MCB University Press, vol. 96, no. 4, 1996, pp. 11–17.

[5]. X. Blanc, A. Mougenot, I. Mounier, T. Mens. Incremental Detection of Model Inconsistencies based on Model Operations. In Advanced Information Systems Engineering, CAiSE 2009, LNCS, vol. 5565, Springer, 2009, pp. 32-46.

[6]. C. Xu, C.S. Cheung, W.K. Chan. Incremental Consistency Checking for Pervasive Context. In Proc. the 28th International Conference on Software Engineering, 2006, pp. 292-301.

[7]. J. Harrison, S.W. Dietrich. Towards an Incremental Condition Evaluation Strategy for Active Deductive Databases. In Research and Practical Issues in Databases, World Scientific, 1992, pp. 81-95.

[8]. ISO 10303-11: 2004. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual, ISO, 2004.