7-1-2013

Constraint-Based Data Quality Management Framework For Object Databases

David Weber
ETH, Zurich, Switzerland, weber@inf.ethz.ch

Stefania Leone
ETH, Zurich, Switzerland, leone@inf.ethz.ch

Moira Norrie
ETH, Zurich, Switzerland, norrie@inf.ethz.ch

CONSTRAINT-BASED DATA QUALITY MANAGEMENT FRAMEWORK FOR OBJECT DATABASES

Weber, David, ETH Zurich, Universitaetstrasse 6, 8092 Zurich, Switzerland, weber@inf.ethz.ch

Leone, Stefania, ETH Zurich, Universitaetstrasse 6, 8092 Zurich, Switzerland, leone@inf.ethz.ch

Norrie, Moira C., ETH Zurich, Universitaetstrasse 6, 8092 Zurich, Switzerland, norrie@inf.ethz.ch

Abstract

When developing large information systems, several forms of data quality checks and controls have to be specified. However, today's object databases only provide very limited support for constraint definition, which causes software engineers to implement data quality controls such as consistency checks and exception handling in the application code. Given that it is common for several applications to access the same database, constraints may even be distributed across applications. As a consequence, the same or similar constraints may be checked multiple times in different components of an information system, or even in different applications, which increases code redundancy. Changing constraints may affect multiple parts of the application or even multiple applications.

We therefore propose an approach that complements object databases with support for constraint definition, management and validation. We introduce a framework where data quality controls are specified by means of constraints which are defined and managed within the framework and validated against the database based on an event-condition-action paradigm. Our approach provides flexible and customisable constraint validations, where constraints can either be used to ensure data quality in a strict and traditional way with constraint violations resulting in exceptions, or a relaxed way where constraint violation will generate user warnings and indicate possible deficiencies in data quality. Our framework is extensible and developers are free to introduce new types of constraints. We present our implementation for the object database db4o.

Keywords: Data Quality Management, Constraints, Object Databases, Framework, db4o.

1 Introduction

While constraints are regarded as a standard approach for ensuring basic data quality in terms of data integrity and consistency, today's object databases still provide very limited support for constraint definitions. For example, the object databases db4o[1] and Versant[2] only provide support for unique field value constraints on single object attributes.

As a consequence, software engineers are forced to implement data quality controls in the application code. Constraint definitions, such as basic checks for type safety of an input field in a graphical user interface as well as complex business rules, are distributed across the information system and data quality checks are performed in different parts of the system.

This makes it difficult for a developer to keep an overview of all data quality controls and to manage them efficiently. Furthermore, there is often redundancy as the same checks may be defined and executed in different layers of an information system or even in different client applications, resulting in increased maintenance efforts and reduced performance. To make things worse, information systems are usually not just linear architectures where only one application accesses a database. It is more likely that several applications access a shared database and that the information system exchanges data with other applications via machine-to-machine interfaces. Consequently, identical constraints might not only be checked multiple times in different layers of the information system but even by different client applications. Thus, it would be very helpful for software engineers if they could build upon a framework that extends the capabilities of today's object databases with support for constraint definition, management and validation in a centralised component.

In this paper, we present an approach, framework and implementation of a constraint-based approach to data quality that extends and complements object databases with support for constraint-based data quality management. With our approach, information systems are designed by means of a semantically rich data model, from which constraints are extracted and imported into our constraints framework. Users are free to configure additional constraint definitions to those extracted. The constraints are managed within our framework and validated against the database based on an event-condition-action (ECA) paradigm.

While constraint violations are typically prevented by an action that, for example, aborts a transaction or throws an error message, in our approach, such actions may be configured to perform arbitrary functionality. For instance, there may be cases where data quality violations should be handled in a more relaxed way such as simply informing the user of possible data quality deficiencies.

Our framework is extensible and developers are free to define new custom constraint types. Moreover, the fact that constraints are managed in a separate component, independent of the application code, allows for the reuse of constraint definitions across applications. This further reduces redundancies and increases development efficiency.

Section 2 presents the background of our work. In Section 3, we introduce our constraint-based approach to data quality management. The constraints framework is presented in Section 4, while we describe in Section 5 how our framework can be extended with new constraint types. Section 6 presents the implementation of the framework and Section 7 introduces the scenario application used to evaluate the framework. Finally, further steps are discussed in the conclusions.

[1] http://www.db4o.com/
[2] http://www.versant.com/

2 Background

A lot of research has been conducted in the field of semantic data models, for example early efforts include (Hammer and McLeod, 1978), (Peckham and Maryanski, 1988) and (Norrie, 1993) where data models with semantically rich constructs and forms of constraints were proposed. These models are targeted at capturing the semantics of an application domain in a more natural way. For example, in (Norrie, 1993), an object-oriented model is proposed that integrates features of extended ER models such as rich classification structures over collections of objects and relationships.

Several approaches of embedding constraints into object databases have been introduced. In (Narasimhan et al., 1994) they provide simple error check methods for individual classes as well as the organisation of integrity constraints into duplicated classes. In (Elmasri et al., 1993) and (Bouzeghoub and Métais, 1991), they propose constraint checking in basic update operations. More advanced approaches are COI methods (Fahrner and Vossen, 1995) where integrity constraints are declaratively stored alongside each class using attributes and DICE (Fahrner et al., 1997) where every integrity constraint type is implemented by a parametrised integrity check method and an exception handling method.

Nonetheless, such approaches have not been applied in practice and are not integrated in current object databases. In fact, today's databases generally lack sophisticated constraint support. RDBMSs typically provide support for integrity constraints, such as key and domain constraints. *Inherent* integrity constraints are represented with the DDL at schema definition time and *explicit* constraints are expressed and enforced by verification mechanisms such as check conditions and assertions. Today's object databases provide even less constraint support. For example, db4o and Versant only provide unique field constraints and explicit constraints are usually specified by means of methods (Beneventano et al., 1998). For constraint enforcement, typically active database techniques such as the event-condition-action (ECA) paradigm are used (Paton and Díaz, 1999). The checking of integrity constraints in object databases is a fundamental problem in database design (Formica, 2002) (Rao, 1994) and the functionality to declare, enforce and maintain integrity constraints in existing object databases is still very limited (Eick and Werstein, 1993) (Zaqaibeh and Daoud, 2008). Furthermore, both database management system types lack functionality for defining more complex constraints or relaxed validations of data quality controls, which, for example, only indicate possible data quality deficiency.

Other approaches have been proposed to overcome the distribution of constraints and consequent redundancies as well as facilitating constraint management. So-called centralised integrity maintenance approaches (Zaqaibeh and Daoud, 2008) (Do et al., 1997) (Urban and Desiderio, 1992) (Eick and Werstein, 1993) introduce a dedicated component next to the database that centralises the management of integrity constraints and is responsible for constraint enforcement. We build on this idea and provide a framework specifically targeted at enriching object databases with rich constraint-based data quality management. Defining and managing constraints separately from the object database allows the definition of complex and sophisticated constraint types which may go beyond the constraint support proposed for object databases, for example (Fahrner and Vossen, 1995) and (Fahrner et al., 1997). In addition, our approach introduces relaxed constraint validation, where violations may result in a user-defined action. To allow for the definition of arbitrary data quality controls and complex business rules, our framework is extensible and new types of constraints can be implemented.

For constraint definition, a number of declarative constraint definition languages have been proposed, for example (Urban and Wang, 1995) and (Bailey et al., 2002), which support the specification of active database rules with an object-oriented view of data. In software engineering, the Object

Constraint Language (OCL)[3] (Redman, 1997) is used to define constraints for UML models (Rodríguez-García et al., 2010). We take a similar approach, where constraints can be defined declaratively by the user alongside the data model.

3 Approach

It is a complex undertaking to ensure data quality in large applications. Our approach supports a software developer in defining, enforcing and maintaining constraints in information systems in order to enhance and control the data quality. The goal of our approach is to provide the uniform handling of different constraint types. Therefore, we separate the constraint definitions and validations of a given application domain from its database, business logic and graphical user interface and propose a framework that manages and validates constraints in a clearly represented and well organised manner. Our focus lies on managing constraints for semantically rich data models in order to complement object databases with an extended data quality management facility.

Our approach distinguishes two types of constraints: hard constraints and soft constraints. *Hard constraints* such as integrity constraints are necessary conditions that data has to fulfil to be valid. If a hard constraint is violated, data quality is deficient. For example, a hard constraint could specify that an attribute value for email address has to contain an @ in order to be valid. To offer more flexibility, *soft constraints* are constraints that are validated in a more relaxed way compared to hard constraints. They could also be seen as recommendations since they are not just valid or invalid, but rather increase or decrease data quality. For example, an attribute description could be recommended to be longer than 30 characters to increase data quality. Another example is the age of data, which could indicate that data is outdated. Violations of soft constraints do not lead to errors, but the user is instead informed of possible data quality deficiencies.

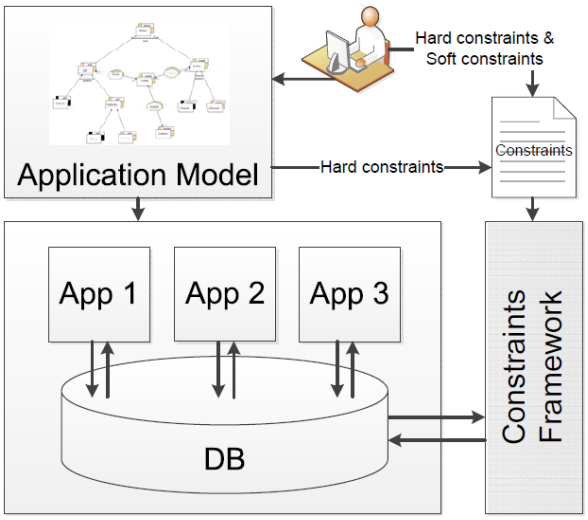

Figure 1. Approach Overview

As shown in the top section of *Figure 1*, the design of an application is divided into the design of the structural data model and data quality controls represented by means of constraints. In our approach, data quality controls consist of inherent hard constraints that are extracted from the application model into constraint definitions and additional explicit constraints may be defined by the developer. Such explicit constraints can either be hard or soft constraints. As shown in *Figure 1* on the right side, these

[3] http://www.omg.org/spec/OCL/2.2/

constraint definitions are taken as input to our constraints framework which manages and validates the defined constraints while the applications are running.

As indicated in *Figure 1*, our approach allows the definition of constraints for various applications that access the same database and it is possible to reuse constraint definitions from one application in another. For example, a course management application running on top of a university database could reuse constraints defined in a study management application running on top of the same database. The constraints framework interacts with the database based on an ECA paradigm, where constraint checks are triggered by database events. In the case of a violation of a hard constraint, our framework triggers an exception in the application, while a warning is thrown in the case of a soft constraint violation.

4 Constraints Framework

In this section, we present the constraints framework which realises our approach and is used to define, validate and manage constraints.

The constraints for an application are defined with a constraint definition language that is based on the DDL proposed in (Würgler, 2000). Our approach supports the definition of simple constraints such as consistency constraints for single attributes, but it is also possible to define constraints that involve more than one object. We distinguish the definition of hard constraints and soft constraints with the appropriate key words *hardconstraint* and *softconstraint*.

The constraint definitions can have different structures depending on the constraint type but are always of the form of

```
type name details [events] validator {'message'};
```

The type defines if the constraint is validated as a hard or soft constraint. Each constraint has a name which identifies the kind of constraint and details which provide information about the individual attributes and conditions. The details are followed by a list of database events upon which the validation should take place. The validator defines the validation type. This can be the standard validator or a custom one. The standard validator offers simple attribute validations, while the custom validators usually offer more complex validations. The message is optional, and, if present, it will be used for the exception or text message, respectively. Otherwise, a standard text will be used. Below, we give an excerpt of a constraint definition file with four constraint definitions.

```
hardconstraint length Address.phone = 13 [Creating, Updating] Standard;
softconstraint length Course.description >=30 [Creating, Updating] Standard;
hardconstraint association attends from Student (5:*) to Course (0:*) [Committing] Custom;
hardconstraint cover (Staff and Student) Person [Committing] Custom 'Cover violated';
```

As shown in the example above, the *length* constraint could be defined as a hard or soft constraint. The first constraint definition defines that the attribute `phone` of the class `Address` must have the length of 13. This will be checked upon the database events `Creating` and `Updating`, thus when an `Address` object is created or updated and the object is to be stored in the database. The validation will be done with the standard validator and, if a violation occurs, an error with the default message for the length constraint will be thrown. The `length` constraints are instances of two different constraint types with the same constraint name but different validations. The default message is of the form 'Constraint violated:' with an output of the constraint details attached as a string. The second line also defines a length constraint specifying that the attribute description of the class `Course` should be at least 30 characters long. If a violation occurs only a message will be thrown as a hint that the data quality could be increased by adding more details to the description. On the third and fourth line, more

complex constraints are defined: an *association* with cardinality constraints and a *cover* constraint specifying that every person must be either staff or student (or both). Both definitions are validated with a custom validator upon the `Committing` event. The cover constraint defines a custom message which is passed to the application upon constraint violation.

AssociationConstraint
String associationName; String collectionSource; String collectionTarget; String minSourceCardinality; String maxSourceCardinality; String minTargetCardinality; String maxTargetCardinality; boolean customValidate(ObjectContainer database, Object objectToValidate);

Figure 2. Association Constraint

The constraint types are represented by Java[4] classes defining particular attributes that hold the detailed information. As an example, the UML diagram of an association constraint is depicted in *Figure 2*. It shows the attributes that are required to store the detailed information of the association constraint and the `customValidate` method.

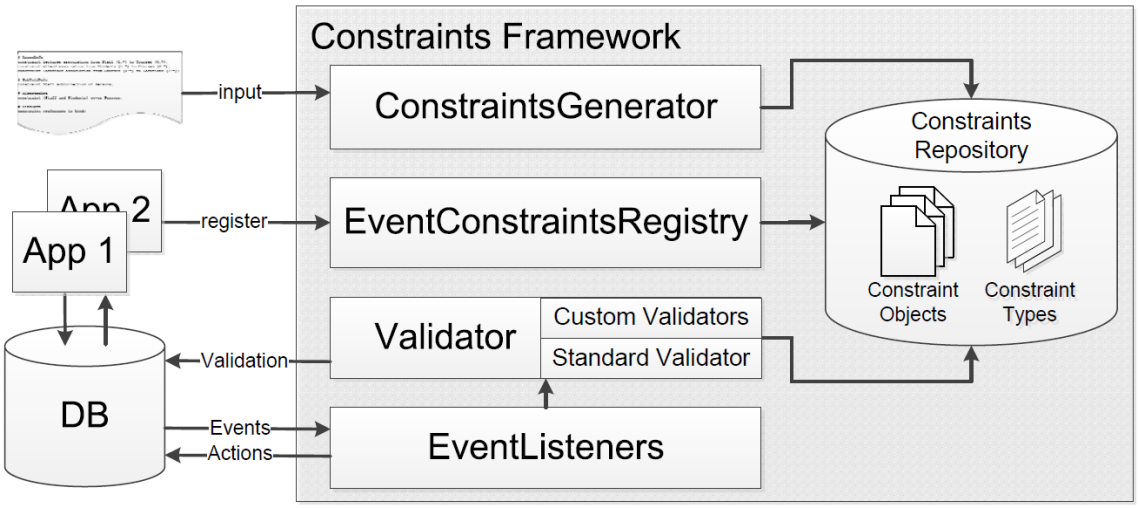

Figure 3. Constraints Framework

Figure 3 gives an overview of the constraints framework and its different components. In the left upper corner, the input file is depicted that contains all constraint definitions. The ConstraintsGenerator is responsible for parsing the input file and generating the appropriate constraint definition objects and validation configuration. The constraint definition objects and the validation configuration are stored in the constraints repository. The EventConstraintsRegistry is responsible for linking the constraints from the repository to the appropriate EventListeners depending on the database events that were configured with the constraints in the input file. The EventConstraintsRegistry also adds the EventListeners as listeners to the adequate db4o database events. For example, an association that is defined with the `Creating` event and custom validation would cause the framework to associate the 'CreatingListener' and the concrete association constraint

[4] http://www.java.com/

to it. Afterwards, the 'CreatingListener' would be added as a listener to the `Creating` event. Upon notification, the 'CreatingListener' would trigger the custom validation.

The method `EventConstraintRegistry.registerForValidation(DB)`is invoked by applications to register themselves and their database, `DB`, with the constraints framework.

The framework works with an event-condition-action mechanism. If a database event occurs due to database operations triggered by the applications, the applicable EventListener is notified. The EventListener holds a list of constraints that were configured to be validated with the according event. Upon a triggered event, the suitable EventListener passes the constraint list with the associated objects for validation to the Validator. The Validator decides which validation method has to be invoked for each constraint based on the configured validation type in the constraint and consequently invokes the validation of the constraint with the objects received from the database event. As a standard validator, we use the validator from the validation framework OVal[5]. OVal is a pragmatic and extensible general purpose validation framework for Java where objects can easily be validated on demand. To validate objects, it extracts a validation configuration from the constraints repository. As shown in *Figure 3*, the validations can imply the participating objects received from the db4o event but validations can also perform extensive checks against the database.
Depending on the validation result, an exception or a message is thrown, and the transaction may be aborted. Hard constraints will rather raise an exception while soft constraints will alert with a message. The application then has to handle the validation result of the framework.

5 Extensibility

Our framework is extensible and developers are free to introduce new constraint types. New constraint types are defined by means of a user-defined constraint class that describes the constraint with its particular attributes and one or more validators.

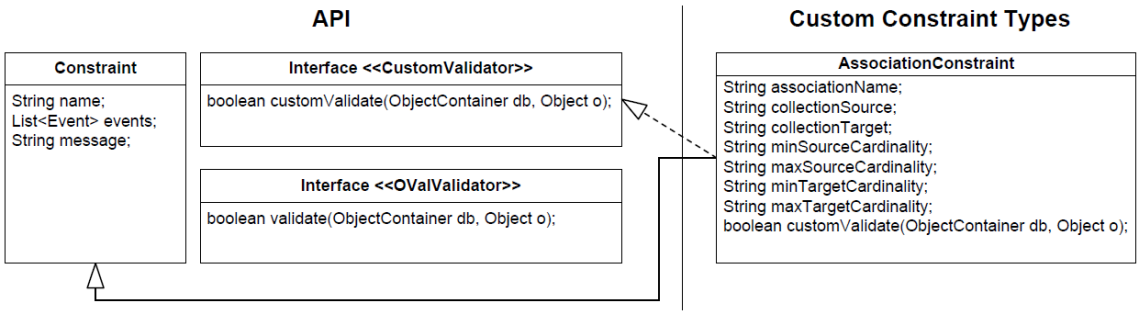

Figure 4. Constraint Types

A user-defined constraint class extends the `Constraint` class as shown in *Figure 4*. A constraint has a name, defines a list of events specifying when it will be checked and a message that may be used upon constraint violation. On the right of *Figure 4*, the `AssociationConstraint` class is shown as an example of a custom constraint defining cardinality constraints.

Depending on the desired validation type, the custom constraint class implements one or both of the interfaces `CustomValidator` and `OValValidator` depicted in *Figure 4*. The methods `validate` and `customValidate` respectively, will be invoked by the validator during the constraint

[5] http://oval.sourceforge.net/

validation. The `validate` method defines a validation based on OVal, while the method `customValidate` may define any user-defined validation.

An excerpt of the custom validation for a maximum length constraint is depicted below:

```
    public boolean customValidate(ObjectContainer db, Object objectToValidate) {
        boolean valid = true;
        int length = getLength(objectToValidate, this.classToValidate, this.attributeName);

        if (length > this.maxLength) {
            valid = false;
        }

        return valid;
    }
```

The code snippet shows the `customValidate` method of a custom maximum length validation which checks if an attribute of an object exceeds a certain length. The `objectToValidate` is the object that is to be validated. The reference to the database `db` is not needed here. The `classToValidate`, `attributeName` and `maxLength` are particular constraint attributes which define the class, the attribute and the maximum length that should be validated. In the `getLength` method, a class cast for the `objectToValidate` would be done with the `classToValidate` and the length of the attribute with the `attributeName` would be evaluated. In the `customValidate` method, it is then determined whether the length exceeds the maximum length.

In a second step, the parser has to be extended in order to translate declarative constraint definitions specified in the input file into constraint definition objects that are used by the framework at run-time. As an example, a part of the association parser configuration is shown below:

```
    /* Association Definition */
    <ASSOCIATION>
    associationName = <IDENT>
    <FROM>
    collectionSource = <IDENT>
    <OPAR>
    ( minSourceCardinality = <NUMBER> | minSourceCardinality = <STAR> )
    <COLON>
    ( maxSourceCardinality = <NUMBER> | maxSourceCardinality = <STAR> )
    <CPAR>
    <TO>
    …
    {
      return new AssociationConstraint(
               "Association Definition",
               associationnName.image,
               collectionSource.image,
               minSourceCardinality.image,
               maxSourceCardinality.image,
               …);
}
```

In the first block, the code snippet shows the read-out of the specific constraint attributes according to the grammar specification of the parser. In the second block, the constraint object instance `AssociationConstraint` is instantiated.

Our framework also allows the definition of new EventListeners. Currently, our framework offers EventListeners for all events of db4o. If the framework was to be used with another database which offers additional event types, developers have the possibility to implement custom EventListeners.

Such a custom EventListener is a subclass of our abstract `EventListener` class and implements the db4o `EventListener4` interface which defines an `onEvent` method, as shown below, that takes an `eventType` and, depending on that event type, a list of event type arguments. For example, the db4o

update event `Updating`, which is fired before an object is updated, takes a `CancellableObjectEventArgs` object as argument, through which the updated object can be accessed and the transaction can be aborted.

```
public void onEvent(EventType<...> eventType, List<EventTypeArguments> args) {…}
```

Finally, new EventListeners have to be registered together with the respective database event in the `EventConstraintsRegistry`.

6 Implementation

The implementation of our framework consists of general framework components and integration with db4o as shown in *Figure 5*.

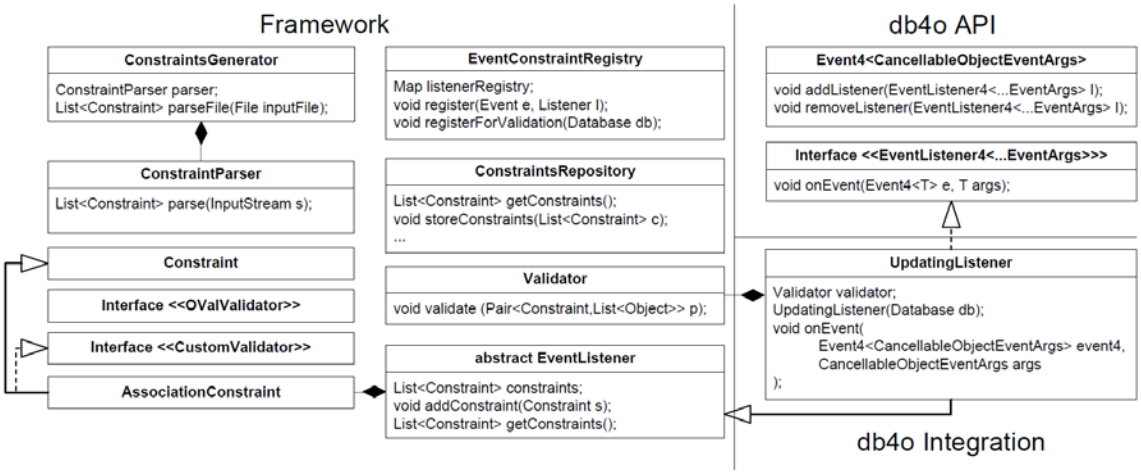

Figure 5. Framework

The `ConstraintsGenerator` in the upper left corner is responsible for parsing the constraint definitions by making use of the `ConstraintParser` that was generated from the constraint definitions shown in Section 5 with JavaCC[6]. The `ConstraintsGenerator` initialises the constraint objects using concrete constraint classes, as exemplified by the `AssociationConstraint` class in the bottom left corner of *Figure 5*, and stores them in the `ConstraintsRepository`. Currently, we support basic integrity constraints such as type checks as well as rich semantic constraints as defined in (Norrie, 1993).

At run-time, applications may register their database with the `EventConstraintRegistry`, which is then responsible for loading all constraints defined for that application from the `ConstraintsRepository`. It also manages a mapping between EventListeners and events which is store in the `EventConstraintRegistry.listenerRegistry` map. As already mentioned, our framework could be used with any object database that offers access to database events based on an ECA paradigm. For each event type offered by the object database, an EventListener has to be implemented. For our implementation, we used the db4o event registry API which gives access to all db4o events[7]. In *Figure 5*, we illustrate the `UpdatingListener` class, which is an `EventListener`

[6] http://javacc.java.net/
[7] http://community.versant.com/documentation/Reference/db4o-8.1/java/reference/

for the `Updating` event. It is a subclass of `EventListener`, implements the db4o `EventListener4` interface and holds a reference to the database and a list of constraints that were configured to be validated with that specific event. When an `Updating` event occurs the `UpdatingListener` is notified by invoking its `onEvent` method with the db4o event and the appropriate event arguments, e.g. the object that has to be validated. The `UpdatingListener` passes the constraint list with the associated objects for validation to the `Validator` by invoking its `validate` method. The `Validator` decides which validation method has to be invoked for each constraint based on the configured validation type in the constraint and consequently invokes the appropriate validation method of the constraint. Depending on the validation result, an exception or a message is thrown, and the transaction may be aborted.

7 Evaluation

To validate our approach, we developed two applications for a university database that manages staff, students and courses, as shown in the left upper corner of *Figure 6*. The student application is used by students to manage their studies and register for courses, while the lecturer application is used by lecturers to manage their courses.

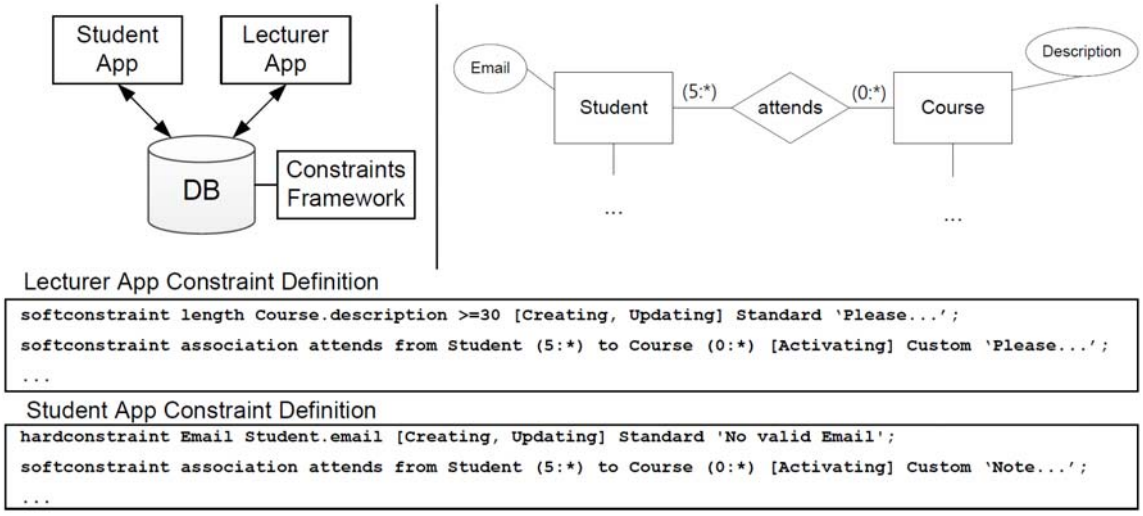

Figure 6. Applications

On the right of *Figure 6*, we show an excerpt of the data model, describing that students may attend courses. The two applications share the same data model, but define their own constraint definitions, based on the performed application logic, as shown at the bottom of *Figure 6*.

For the lecturer application, a violation of the length constraint on `Course.description` will result in a message of the form 'Please provide a more detailed description, if possible.' which, in our lecturer application, is propagated and displayed to the user in a text box, and the user is free to react.

The association constraint is defined by both applications for the `Activating` event, which is triggered when data is accessed. The constraint definitions only differ in the specified message. As long as fewer than five students have registered, when accessing the course site through the lecturer application, the lecturer is presented with a message stating that fewer than 5 students have registered so far. Then, when a student accesses the courses they plan to attend through the student application, a message is displayed stating that the course may not take place, because less than 5 students have registered.

When a student enters an invalid email address, the framework will throw a validation exception and the transaction is aborted. The error message is also propagated to the user interface and the user is presented with a message that the entered email address is not valid.

8 Discussion and Conclusions

We have presented an approach, framework and implementation of a constraint-based approach to data quality management for object databases, in particular proposing a solution for the object database db4o. Our framework, however, could also be used for other object databases and we also plan to support the relational model to provide advanced support for data quality validation based on constraints. Our approach therefore is designed to offer an independent framework rather than as an integral part of a database.

We currently provide support for basic constraint checks as offered by OVal as well as rich classification constraints. We plan to extend the framework with constraint types that allow the definition of complex data quality controls that cover data quality dimensions such as the ones defined in (Wang and Strong, 1996) (Wand and Wang, 1996). Such an extension would also identify for which dimensions constraints can be implemented and for which dimensions it would be difficult or even impossible. For example, we can think of defining constraints for dimensions such as timeliness and completeness but it seems to be difficult to define constraints for other dimensions like reliability.
Since the implementation of a comprehensive set of constraints types covering all these data quality dimensions is still work in progress, we do not have a complete formal definition of the constraints language. Our definition language is based on the OMS DDL presented in (Würgler, 2000) and the complete formal definition is a goal of future work.

Furthermore, while our current implementation supports the definition and validation of hard and soft constraints, we plan to generalise this approach so that arbitrary actions can be performed upon a constraint violation.

References

Bailey, J., Poulovassilis, A. and Wood, P. T. (2002). An event-condition-action language for XML. in *Proceedings of the 11th international conference on World Wide Web*, Honolulu, Hawaii, USA, 511509: ACM, 486-495.

Beneventano, D., Bergamaschi, S., Lodi, S. and Sartori, C. (1998). Consistency Checking in Complex Object Database Schemata with Integrity Constraints. *IEEE Transactions on Knowledge and Data Engineering.* 10, 576-598.

Bouzeghoub, M. and Métais, E. (1991). Semantic Modeling of Object Oriented Databases. in *Proceedings of the 17th International Conference on Very Large Data Bases*, 672308: Morgan Kaufmann Publishers Inc., 3-14.

Do, N. C., Bae, S. M. and Choi, I. J. (1997). Constraint maintenance in engineering design system: An active object-oriented approach. *Computers & Industrial Engineering.* 33, 643-647.

Eick, C. F. and Werstein, P. (1993). Rule-based consistency enforcement for knowledge-based systems. *Knowledge and Data Engineering, IEEE Transactions on.* 5, 52 -64.

Elmasri, R., James, S. and Kouramajian, V. (1993). Automatic class and method generation for object-oriented databases in Ceri, S., Tanaka, K. and Tsur, S., eds., *Deductive and Object-Oriented Databases*, Springer Berlin Heidelberg, 395-414.

Fahrner, C., Marx, T. and Philippi, S. (1997). DICE: declarative integrity constraint embedding into the object database standard ODMG-93. *Data Knowl. Eng.* 23(2), 119-145.

Fahrner, C. and Vossen, G. (1995). A survey of database design transformations based on the Entity-Relationship model. *Data & Knowledge Engineering.* 15(3), 213-250.

Formica, A. (2002). Finite satisfiability of integrity constraints in object-oriented database schemas *Knowledge and Data Engineering, IEEE Transactions on.* 14(1), 123-139.

Hammer, M. and McLeod, D. (1978). The semantic data model: a modelling mechanism for data base applications. in *Proceedings of the 1978 ACM SIGMOD international conference on management of data*, Austin, Texas, 509264: ACM, 26-36.

Narasimhan, B., Navathe, S. B. and Jayaraman, S. (1994). On mapping ER and relational models into OO schemas in Elmasri, R., Kouramajian, V. and Thalheim, B., eds., *Entity-Relationship Approach — ER '93*, Springer Berlin Heidelberg, 402-413.

Norrie, M. C. (1993). An extended entity-relationship approach to data management in object-oriented systems in Elmasri, R., Kouramajian, V. and Thalheim, B., eds., *Entity-Relationship Approach — ER '93*, Springer Berlin Heidelberg, 390-401.

Paton, N. W. and Díaz, O. (1999). Active database systems. *ACM Comput. Surv.* 31(1), 63-103.

Peckham, J. and Maryanski, F. (1988). Semantic data models. *ACM Computing Surveys (CSUR).* 20(3), 153-189.

Rao, B. R. (1994). *Object-Oriented Databases: Technology, Applications and Products.* McGraw Hill.

Redman, T. C. (1997). *Data Quality for the Information Age.* Artech House Inc.

Rodríguez-García, D., Barriocanal, E. G., Alonso, S. S. and Nuzzi, C. R.-S. (2010). Defining Software Process Model Constraints with Rules Using OWL and SWRL. *International Journal of Software Engineering and Knowledge Engineering.* 20(4), 533-548.

Urban, S. D. and Desiderio, M. (1992). CONTEXT: A CONstrainT EXplanation Tool. *Data & Knowledge Engineering.* 8, 153 - 183.

Urban, S. D. and Wang, A. M. (1995). The design of a constraint/rule language for an object-oriented data model. *Journal of Systems and Software.* 28, 203 - 224.

Wand, Y. and Wang, R. Y. (1996). Anchoring data quality dimensions in ontological foundations. *Commun. ACM.* 39, 86-95.

Wang, R. Y. and Strong, D. M. (1996). Beyond accuracy: what data quality means to data consumers. *J. Manage. Inf. Syst.* 12, 5-33.

Würgler, A. P. (2000). *OMS development framework.* unpublished thesis ETH Zürich.

Zaqaibeh, B. and Daoud, E. A. (2008). The Constraints of Object-Oriented Databases. *International Journal of Open Problems in Computer Science and Mathematics (IJOPCM).* 1, 11-17.