An Approach to the Validation of XML Documents Based on the Model Driven Architecture and the Object Constraint Language

Denis A. Nikiforov(✉), Dmitriy V. Korj, and Ruslan L. Sivakov

Center of Information Technologies LLC, Ekaterinburg, Russia
{Denis.Nikiforov,Dmitriy.Korj,Ruslan.Sivakov}@centre-it.com

Abstract. It is possible to develop data processing applications using a variety of different data representation formats (EDI, CSV, XML, JSON), domain-specific languages, and general-purpose programming languages (XSLT, SQL, Java, C#). On the one hand, such a variety allows one to choose the most optimal data format or language based on the specific requirements being applied, while on the other one, contemporary information systems or complexes of integrated information systems have become similar to the Tower of Babel, being so cumbersome to build and maintain. A possible solution to this issue could be found in developing platform-independent specifications to be used for generating the source code for each required platform.

This article describes an approach to the XML document validators' generation based on UML models with Object Constraint Language (OCL) rules. The authors give a brief account of similar tools and propose a generalized schema for generating the validators based on a model-driven approach. The core component of this schema is the transformation of OCL constraints to XPath assertions. The first ones could come from one of the supported platform-independent models (Eurasian Economic Union Data Model or ISO 20022), while the later could be embedded into XML Schema 1.1, XSLT or Java code.

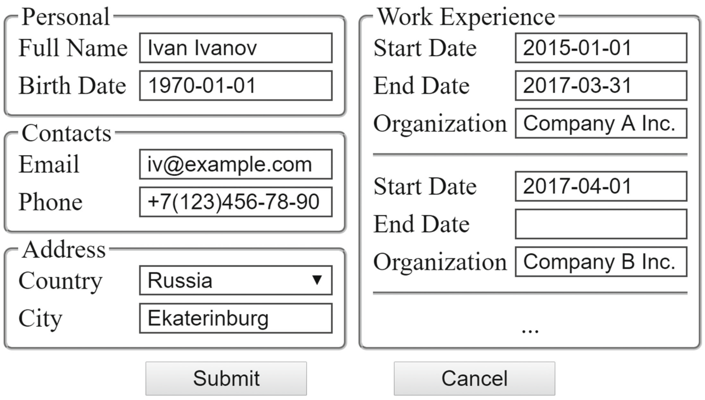
The transformation is implemented at the model level in the Query/View/Transformation language. The article does not go into details of converting OCL into XPath, because such a description takes up a lot of space and has already been given in similar articles. The authors describe only the key features of their approach: development of metamodels for XPath, XSD 1.1, and XSLT, support of a variety of platform-independent and platform-specific models, determination of elements subject to validation, external data sources, kinds of validation messages, preconditions.

Keywords: XML validation · Semantic validation
Unified Modeling Language · Object Constraint Language
XPath · XML Schema · XSLT · Model Driven Architecture
Platform-independent model · Metamodel · Model transformation
Query/View/Transformation · Eclipse Modeling Framework
ISO 20022

A. K. Petrenko and A. Voronkov (Eds.): PSI 2017, LNCS 10742, pp. 291–305, 2018.
https://doi.org/10.1007/978-3-319-74313-4_21

1 Introduction

Consider the following example. Let us say you are designing a website where users can post their resumes (Fig. 1). Among other things, users are to specify an employment period for each job they had, however they only specify a starting date for their currently held job. You could introduce the following constraint to ensure consistency of each job's dates: if an end date is specified, then it must not be earlier than the starting date. Such a restriction can be easily implemented in JavaScript.



Personal

Full Name	Ivan Ivanov
Birth Date	1970-01-01

Contacts

Email	iv@example.com
Phone	+7(123)456-78-90

Address

Country	Russia ▼
City	Ekaterinburg

Work Experience

Start Date	2015-01-01
End Date	2017-03-31
Organization	Company A Inc.
Start Date	2017-04-01
End Date	
Organization	Company B Inc.

...

Fig. 1. Example of a data-entry form for resume details

Subsequently, you decide to make it easier for users to add their resumes to your site, and implement an option allowing them to upload the resumes in an XML format:

```
<CurriculumVitae>
  <!-- ... -->
  <WorkExperience><!-- Past Employment -->
    <StartDate>2015-01-01</StartDate>
    <EndDate>2017-03-31</EndDate>
    <Organization>Company A Inc.</Organization>
  </WorkExperience>
  <WorkExperience><!-- Current Employment -->
    <StartDate>2017-04-01</StartDate>
    <Organization>Company B Inc.</Organization>
  </WorkExperience>
</CurriculumVitae>
```

You can use an existing JavaScript code to validate the XML documents if the server-side logic is implemented on Node.js, however this is not always the case, and you may need to do a repeat implementation of the constraints by means of XSD 1.1, XSLT, or a general-purpose programming language, such as Java.

Let us assume that despite the implemented constraints your application's database still receives inconsistent data. In order to ensure the data's integrity, you define the constraints in a third language – SQL. This duplication may result in a disagreement of implementations and an increase in the application's development and maintenance costs.

The Object Management Group (OMG) proposes to use the Model Driven Architecture (MDA) to solve this problem [15]. Towards this end, you would define the data schema and constraints in a Platform-Independent Model (PIM), e.g. in UML [28] and OCL [19,26] languages, correspondingly. The constraint under consideration could be defined in OCL in the following manner:

```
WorkExperience->forAll(x |
  x.EndDate->notEmpty() implies x.StartDate <= x.EndDate)
```

You transform the PIM into various Platform-Specific Models (PSM) whenever the need to implement constraints for a particular platform arises, those PSMs being used as a basis for generating the required source code [1,5,16].

This article discusses only the transformations from PIM to PSM, used to generate validators for XML documents. Implementation of constraints for web forms or relational databases lies outside this article's scope.

The paper is organized as follows. In Sect. 2, we define and compare the existing XML document validation tools using OCL constraints. In Sect. 3, we propose a generalized framework for generating validators based on the MDA. In Sect. 4, we define some special aspects of our approach that distinguish it from its analogues. In Sect. 5, we list areas to focus on in the future.

2 Overview of Analogues

There are two approaches to validating XML documents using OCL rules: (1) transforming the rules into expressions in a language that can be run on a certain technology platform (Schematron [10], XPath, Java, etc.) or (2) interpreting the rules within the context of an XML document's object model [25].

In the first case, the OCL rules get adapted to run on a desired technology platform (Fig. 2), while in the second case it is the data that gets adapted from its platform-specific representation to a suitable platform-independent form (Fig. 3).

The first approach is preferred in situations where the environment, in which the XML documents are to be validated, has to comply with stringent requirements. However, if the part played by overhead costs associated with generating an object model of the XML document is insignificant, and there are no restrictions on the choice of technologies, it is more appropriate to use the second approach. This is due to the fact that generating an object model for data is much simpler than mapping all the operations defined in the OCL Standard Library.

Table 1 describes the existing tools for XML documents validation using OCL rules. The OCL specification's support provided by tools based on the OCL interpreter is more complete. However, the most recent versions of OCLE and

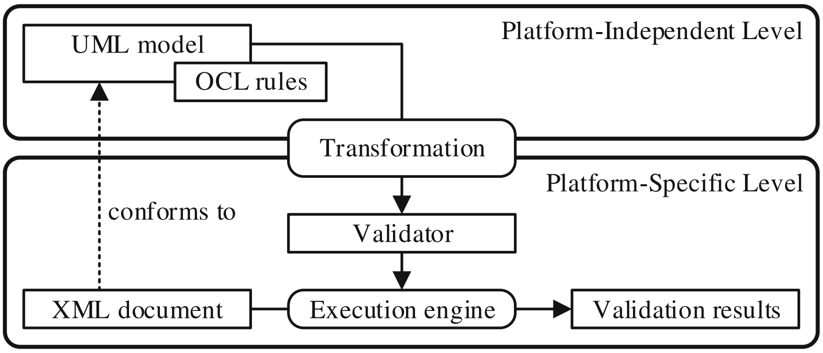


Fig. 2. Flowchart of transformation of OCL constraints

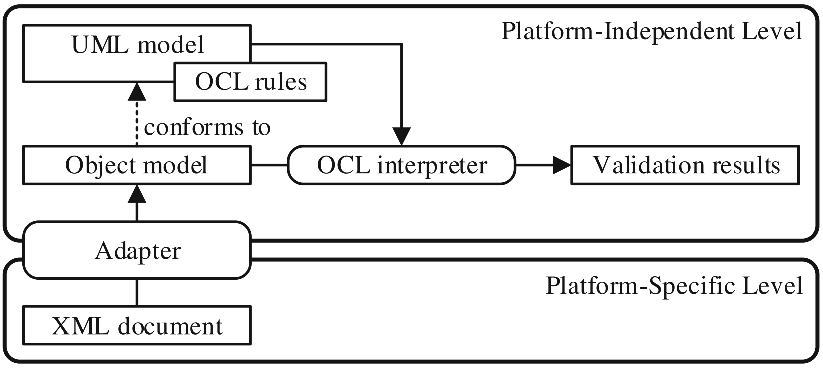


Fig. 3. Flowchart of interpretation of OCL constraints

Dresden OCL date back, respectively, to years 2005 and 2015, meaning that they implement older versions of the specification. Unlike other Schematron-generating tools, eXolutio supports relatively complex “iterate” and “closure” operations, however it is not clear if “substring” is supported [13].

eXolutio and OCLE support plain UML class diagrams as a PIM. ShapeChange and NIEM transformation support UML models as per ISO 19109 and NIEM specification, respectively. For Eclipse OCL, either a simple UML class diagram or an Ecore model adapted for the Pivot Meta-Model can be used as a PIM. Dresden OCL additionally provides adapters for Java classes and XML schemas.

Some tools allow to extend the OCL Standard Library by using custom UML classes with OCL operations. The latter are either interpreted, or transformed to XSLT, XQuery or Java functions.

External data sources are fully supported only by OCLE. All data sources are defined in a single logical model, and can be referenced from the OCL rules.

For eXolutio, an OCL syntax extension to reference external XML documents is proposed in [13].

Most tools are implemented in Java or other general-purpose programming languages. The only exclusion to this is the NIEM PIM to Schematron transformation, having been implemented in QVTo [30], however even in its case the XPath expressions are generated directly as text, bypassing PSM.

Table 1. Comparison of analogues

Tool	Validation mechanism	OCL support	Source PIM	OCL extension mechanism	Support of external sources	Implementation
Shape-Change[a]	Schematron	Extremely limited	ISO 19109	Absent	Absent	Java
NIEM [27]	Schematron	Limited	NIEM	Absent	Absent	QVTo
eXolutio [11,14]	Schematron	Average	UML class diagram	Present	Proposed	Unknown
Dresden OCL [3]	Interpretation, Java	Potentially complete	Adapted model	Present	Absent	Java, EMF
Eclipse OCL [31]	Interpretation	Complete	Adapted model	Present	Absent	Java, EMF
OCLE [2]	Interpretation, Java	Potentially complete	UML class diagram	Present	Present	Unknown

[a]http://shapechange.net/targets/xsd/extensions/ocl/.

3 Proposed Generalized Validator Generation Scheme

Most of the above-discussed tools are solid applications written in general-purpose programming languages. This significantly complicates understanding of how exactly the UML models and OCL rules are transformed in those tools, and makes it difficult to adjust transformations to support other PIM or to better support the OCL Standard Library.

MDA offers an alternative approach to developing applications, where all source, intermediate and target artifacts are considered to be models within a certain modeling space [4]. A development process can be represented as a sequence of model transformations [15]. Each model must conform to a certain metamodel, and each metamodel should be developed in accordance with a certain meta-metamodel. Each meta-metamodel forms its own modeling space [4].

OMG has developed a number of specifications describing metamodels (UML, OCL), meta-metamodels (MOF), model transformation languages (QVTo), text representation of models (XMI). Eclipse Modeling Project (EMP) [6] implements some of these specifications. There are various specifications and tools for model-based development, however, OMG is the main standardizing organization in this field and offers the most complete set of specifications, while the most complete implementation of the specifications is provided by the EMP.

Figure 4 shows our proposed and implemented generalized scheme of validator generation based on UML models with OCL constraints. Metamodels and transformations developed by us at [21–23] are marked in gray in the figure. All text artifacts belong to the EBNF modeling space. UML models, XSLT models, OCL Abstract Syntax Trees (AST), XPath AST belong to the Ecore [31] modeling space (Ecore is an analog of MOF in EMP). The figure depicts the sequence of transformations for a single pair of PIM and PSM. It is possible to use not only XSLT, but also XSD 1.1 or Java as a PSM.

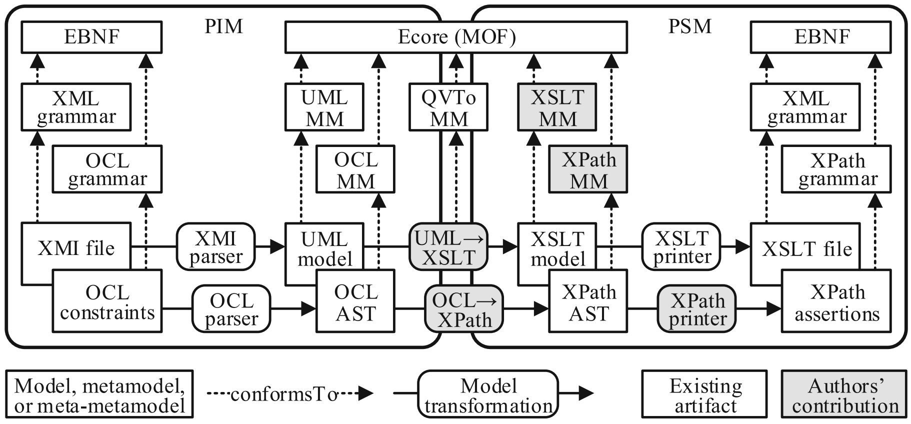


Fig. 4. Proposed generalized validator generation scheme

The transformation is done in three stages. First, the UML model with OCL constraints is translated from a text representation in an EBNF modeling space to an Ecore modeling space [4]. Subsequently, the Ecore modeling space is used to transform PIM to PSM, with the transformation being defined in QVTo. Finally, the PSMs are converted to text and sent back to the EBNF modeling space. Let us review each step in more detail.

3.1 Transformation of PIM from Textual to Ecore Representation

The source UML model is represented in the XMI format [29]. The XMI file contains OCL rules. We use Eclipse UML and Eclipse OCL to parse them [6]. The result is an UML model conforming to an UML metamodel and a set of OCL models (abstract syntax trees) conforming to an OCL metamodel. Both metamodels are based on the Ecore meta-metamodel, so their subsequent transformations can be described in the QVTo language.

3.2 Transformation of PIM to PSM

Currently, two kinds of source PIMs (data model of the Eurasian Economic Union (EAEU)[1] and ISO 20022 [9,12,24]) and three kinds of target PSMs (XML

[1] https://eomi.eaeunion.org.

Schema, XSLT, and Java) are supported by the transformation. A separate QVTo transformation is implemented for each of the six PIM and PSM combinations, while the main part – which is the transformation of OCL to XPath – is reused. In order to implement these transformations via Eclipse QVTo it is necessary to define metamodels based on the Ecore meta-metamodel for each source and target model. For UML and OCL the metamodels are implemented within the EMP.

Moreover, this project includes an implementation of the XSD 1.0 metamodel. We cannot use it in our generator, because it is impossible to embed XPath assertions in XSD 1.0. An XSD 1.1 metamodel is required. The later itself is an XML-based language, it is defined in the corresponding XML schema. We imported this schema with the wizard provided by the Eclipse Modeling Framework (EMF) [17,31], and created an Ecore-based metamodel for XSD 1.1 [21]. Similarly, we have generated on the basis of the XML schema an Ecore-based metamodel for XSLT 2.0 [21].

In addition to XSD and XSLT, our transformation also generates Java code and XPath expressions that are not XML-based languages and that has not XML schemas. Their syntax is defined in EBNF, making it necessary to use other tools to develop Ecore-based metamodels. One of such tools is EMFText [8, 18]. It allows to define a textual representation in an EBNF-like language for Ecore-based metamodels. For Java, an Ecore-based metamodel has already been implemented in the JaMoPP project [7]. For XPath 2.0, we have implemented our own metamodel and described its syntax [22]. General principles of design of Ecore-based metamodels based on EBNF grammars were described by us in [20] using the example of SQL.

3.3 Transformation of PSM from an Ecore Representation to a Textual One

Serialization of Ecore models for XML-based languages (XSD, XSLT) is a trivial task. When importing an Ecore-based metamodel from an XML schema the former is complemented by extended metadata that allow to (de-)serialize Ecore models not just in XMI format, but also in the form of XML documents that conform to the original XML schema. It is possible to generate parsers and printers automatically for languages implemented by means of EMFText. For Java, they are available in the JaMoPP project, for XPath 2.0 – in [22].

3.4 Summary

In a general case scenario, in order to transform n PIMs to m PSMs it is necessary to develop $n \times m$ transformations. In this case, such transformations number $2 \times 3 = 6$. This nonlinear relationship complicates the development of validator generators. The difference between our approach and the approaches described in Sect. 2 is that we have decomposed the transformations as much as possible.

Transformations from a text representation (Sect. 3.1) or into a text representation (Sect. 3.3) are generalized and reusable. Their number is linearly dependent on the number of PIMs and PSMs to be supported by the generator of validators. Some of these transformations had already been implemented (Eclipse XMI parser), other ones we implemented on our own (XPath 2.0 printer) [22].

The transformation from OCL to XPath 2.0 is also generic and does not depend on the PIM and PSM combination. This is described in further details in Sects. 4.1 and 4.2.

It is these transformations that are the most difficult to implement, whereas transformations that are specific to each pair of PIM and PSM are more numerous, but easier to implement.

4 Features Distinguishing Our Approach from Its Analogues

As seen in the case of tools discussed in Sect. 2, the rules of OCL transformation to XPath generally coincide. Therefore, we will not describe them in detail in this paper, especially since the OCL Standard Library defines around two hundred operations. The description of the rules is quite voluminous. Hence, we will describe only those features of our approach that distinguish it from its peers in a fundamental way.

4.1 Different Source PIMs

Each of the analogues described in Sect. 2 can generate validators only for certain kinds of PIM. If you used a different PIM, you would not be able to use any of these tools, or you would have to adapt your PIM somehow. Our generator of validators supports two kinds of PIM (EAEU data model and ISO 20022) and can be easily enhanced to support additional kinds of models.

This is achieved as follows. A significant part of OCL expressions is converted to XPath expressions without regard to which data model they are used in. For example, the trivial “2 + 2 <> 5” expression of OCL is always converted to a “2 + 2 ne 5” XPath expression, regardless of the applicable PIM.

The specifics of various types of PIMs appear only when converting expressions that include calls to object properties (PropertyCallExp). Different PIMs use different UML stereotypes for data elements, data types, attributes, and external data sources. Moreover, their modeling can be done using different UML elements. For example, the data model of EAEU models reusable data elements as classes, while ISO 20022 allows only local elements that are modeled as properties of classes. In order to put aside these differences, the following abstract operations are declared in our transformation: “isDataType”, “isDataElement”, “isAttribute”, “isExternalSource”. These operations have to be implemented to add support for a new PIM.

Moreover, each PIM requires “getQName” and “getUnprefixedQName” operations, that return the model object’s name with and without the namespace prefix, correspondingly, to be implemented. Finally, it is necessary to implement the following operations that allow to abstract the OCL transformation to XPath from primitive type systems which tend to vary from one PIM to another: “isNumericType”, “isStringType”, “isBooleanType”, “isDateType”, “isDateTimeType”, “isTimeType”, “isDurationType”.

The tools discussed in the Sect. 2 have their own private OCL to XPath transformations. Our experience shows that it is possible to develop a generalized transformation treated separately from the original PIM. It is also possible to develop a generalized system of primitive types that unifies the type systems of different PIMs.

4.2 Different Target XPath Host Languages

OCL expressions are meaningful only for some UML or Ecore (MOF) models. Similarly, XPath expression cannot be interpreted by themselves, they are to be embedded into some host language. Our tool supports three host languages: XSD 1.1, XSLT, and Java.

In contrast to the source PIM, the target PSM (host language) does not affect the OCL transformation rules to XPath. Only the upper levels of ASTs of OCL expressions have different ways of transformation. If they contain any “forAll” or “implies” operations, those get transformed to host language expressions, not to XPath expressions. Further details can be found in Sects. 4.4, 4.5, and 4.6.

Table 2 lists the features of XPath host languages that our generator of validators supports. XSD, unlike XSLT and Java, serves for basic structural validation of XML documents. It is much more difficult to check the sequence of XML elements or to make sure there are no unexpected elements through XSLT or Java. Therefore, it is necessary to combine validators based on such technologies with XML schema-based checks.

XML schema 1.1, as opposed to its version 1.0, may be supplemented with XPath assertions that implement relatively complex validation rules. However, in XML schemas the rules are defined for data types, while in the case of using XSLT or Java the XPath processor usually will not be datatype-aware. Hence, if an XML document reuses the same data type for several data elements, one has to duplicate constraints for each element. This is not always a problem, in business applications it is often the elements that are semantically constrained, not the data types, so it is easier to implement the rules using XSLT or Java than XSD.

Many information systems already use XML schemas for validation of XML documents. If XPath assertions generated from the OCL constraints complement these, no further changes to the validator are necessary. Using XSLT or, especially, Java can require significant changes in the execution environment.

However, when using the XML schema, you have almost no influence on the way validation results are presented. XSLT and Java allow to generate different types of messages in the required representation. See more details in Sect. 4.6.

Finally, XML schemas do not allow to validate XML documents using external data source or remote services. For example, they cannot check a code against

Table 2. Outline of host languages

Feature	XSD 1.1	XSLT	Java
Basic structural validation	Yes	No	No
Rules are specified for	Data types	Data elements and data types[a]	Data elements
Execution environment requirements	Low	Moderate	High
Customization of validation results representation	No	Yes	Yes
External data sources and complex constraints	No	Maybe (via extensibility)	Yes

[a]Only for schema-aware XSLT processors.

a code list stored in the database. XSLT (via extensibility) and Java do have the capability to use external data sources.

4.3 External Data Sources

Sometimes you need to check the data contained in an XML document using remote services or external data sources. For example, code values might need to be checked against code lists stored in relational databases. The currently existing tools offer only a very limited support for such data validation rules. Our tool supports two kinds of checks – one with remote and another with local execution of expressions. The first kind corresponds to OCL expressions that include the "allInstances" operation. For example, in the following constraint the database is checked for any resumes with the same telephone number but a different email address:

```
not CurriculumVitae.allInstances()->exists(x |
      x.Email <> self.Email and x.Phone = self.Phone)
```

The body of "exists" iteration is transformed to an XPath expression for its remote execution:

```
fn:not(ext:exists("CurriculumVitae", fn:concat(
         "Email != ", Email, " and Phone = ", Phone)))
```

For simplicity reasons, we do not consider the quoting of parameter values. Many databases support such XPath queries over XML documents.

For the second kind of rules there should be an external data source or service defined in the UML model, e.g. "ExternalService1":

```
ExternalService1::isValidPhone(Address/Country, Phone)
```

A procedure is defined for the service that validates phone numbers against the country indicated in the address. In this case all XPath expressions are computed locally, and only the results are passed to the remote service:

```
ext:call("ExternalService1", "isValidPhone",
         Address/Country, Phone)
```

Validation of a code against a code list can be implemented in either way. In the first case the UML model should define the code list's structure, while in the second it is the interface to access it that has to be defined. Note that XPath and OCL are not intended to describe operations with complex logic. In principle, some constraints can be implemented in these languages, but if such constraints are too complicated, it is advisable to implement them through external services.

4.4 Determination of XML Elements Subject to Validation

OCL constraints are always tied to a specific UML classifier. When checking an OCL constraint, instances of this classifier are considered to be valid or invalid. Accordingly, when converting an OCL constraint to an XPath assertion, fragments of XML documents containing data on the instance are considered valid or invalid.

The validation rules are often specified for the entire message, not for individual data elements. An example of such a rule is the OCL constraint specified in Sect. 1. The following XPath expression would be generated for it:

```
every $x in WorkExperience satisfies
  fn:not($x/EndData) or $x/StartDate <= $x/EndDate
```

If there is at least one entry in the work experience with the starting date falling later than the end date, an error will be issued for the entire resume. XPath, as well as OCL, is an expression language, and it lacks the statements to display error messages. In order to display error messages job-by-job one must expand the "forAll" iteration in the following way:

```
<xsl:for-each select="WorkExperience">
  <xsl:choose>
    <xsl:when test="fn:not(EndData) or StartDate <= EndDate">
      <!-- Success --></xsl:when>
    <xsl:otherwise><!-- Error --></xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

The current context item is the current validation element. For Java or another host language, the "forAll" iteration is expanded in a similar way.

Surely, this will give us more informative validation results, however the area of validation can be narrowed even further. To do this you must find in the OCL rule all sub-expressions that returns values of some data elements. In our case

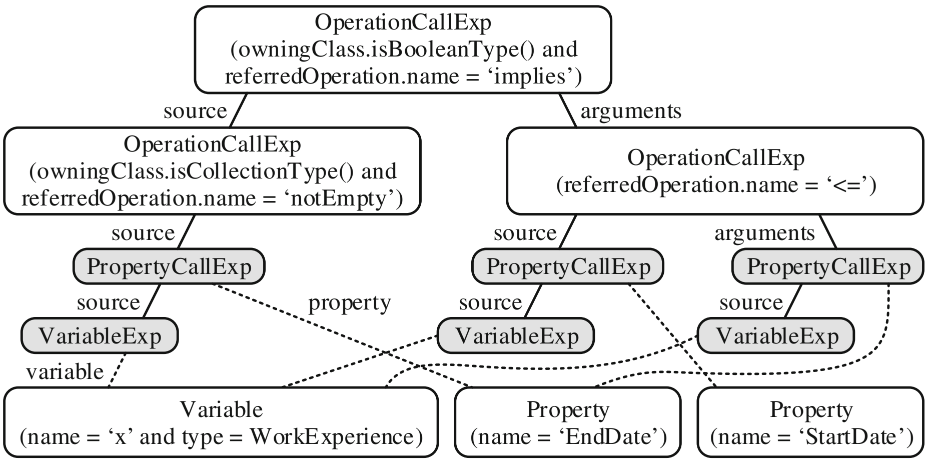


Fig. 5. Example of an abstract syntax tree of an OCL expression

those are the sub-expressions marked gray on Fig. 5: “StartDate” and “EndDate”. In general, it can be not only “PropertyCallExp”, but “IteratorExp”, as well.

4.5 Preconditions

In our example the current job (with a missing “EndDate”) would result in a message declaring that the validation was successful, which is not quite proper, because “StartDate” and “EndDate” were not compared with each other. Often it is practical to distinguish between the following situations: 1) the precondition was met and the test completed successfully; and 2) the precondition was not met and the test was not completed. In such a case, OCL expressions containing the “implies” operation can be expanded as follows:

```
<xsl:choose>
  <xsl:when test="EndData">
    <xsl:choose>
      <xsl:when test="StartDate <= EndDate">
        <!-- Success --></xsl:when>
      <xsl:otherwise><!-- Error --></xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:otherwise><!-- Not checked --></xsl:otherwise>
</xsl:choose>
```

4.6 Kinds of Validation Messages

Usually XML document validation tools allow to specify for each rule the error message that the user sees in case of a data element failing the check. We have implemented five kinds of messages in our tool: success, error, not checked (see Sect. 4.5), not found (see Sect. 4.4), not supported (see Sect. 4.3).

```
<xsl:variable name="__items" as="item()*">
  <xsl:for-each select="WorkExperience">
    <xsl:choose>
      <xsl:when test="EndData">
        <xsl:choose>
          <xsl:when test="StartDate <= EndDate">
            <!-- Success --></xsl:when>
          <xsl:otherwise><!-- Error --></xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:otherwise><!-- Not checked --></xsl:otherwise>
    </xsl:choose>
  <xsl:for-each>
</xsl:variable>
<xsl:copy-of select="$__items" />
<xsl:if test="fn:empty($__items)"><!-- Not found --></xsl:if>
```

Messages may contain embedded OCL expressions:

```
{{StartDate}} must not be greater than {{EndDate}}
```

In general, the embedded expressions can be more sophisticated. Their abstract syntax tree can be analyzed as described in Sect. 4.4, and data elements that served as basis for the computed value can be determined.

5 Conclusion

In Sect. 1 we gave an example of a platform-independent formalization of a constraint in OCL. Section 4.6 presents a fragment of an XSLT code that can be generated based on this constraint using our approach and tools [21–23].

Generating XML document validators based on UML models with OCL constraints cannot be reduced to the transformation of OCL rules into XPath assertions. We have shown in this article that by analyzing the abstract syntax trees of OCL expressions it is possible to generate various subprograms in different host languages (XSLT, Java, etc.) that would test the preconditions, determine elements subject to validation, and output different kinds of validation messages.

Currently, our generator of validators supports only one-third of the OCL Standard Library. This is sufficient for its use in commercial projects. For example, the “closure” iteration that is not supported for the time being, in effect sees very little use in semantic business rules. Subsequently, we plan to implement a full support for the OCL and describe in detail its transformation to XPath.

Our experience shows that OCL allows to specify constraints that are syntactically correct, but make no sense from a business point of view. For example, the rules should be free of preconditions that always return false values or always true values, they should not contain checks for presence of required elements in a document, etc. A number of similar semantic checks of OCL expressions must be performed by the generator. We plan to describe them in more detail.

Further, a formal proof of the validity of OCL to XPath transformation is an important task. None of the existing generators guarantee that the resulting XPath assertions are semantically equivalent to the original OCL constraints.

Finally, another problem pertaining to the development of generators is the lack of a truly platform-independent system of primitive data types. For example, primitive types of ISO 20022 effectively duplicate the XSD data types. That is why the data types are mapped one to one when converting OCL constraints to XPath assertions. On other platforms (SQL, Java, etc.) data types may differ significantly from XSD or ISO 20022, making it difficult to generate code for these platforms. In fact, ISO 20022 is tied to a single platform – XML.

References

1. Cabot, J., Teniente, E.: Constraint support in MDA tools: a survey. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 256–267. Springer, Heidelberg (2006). https://doi.org/10.1007/11787044_20
2. Chiorean, D., Bortes, M., Corutiu, D.: Object constraint language environment, a tool supporting teaching and learning UML and OCL, the understanding and using of metamodeling, abstraction and design by contract. In: Eight Workshop on Pedagogies and Tools for Teaching and Learning Object Oriented Concepts (2000)
3. Demuth, B., Hussmann, H., Loecher, S.: OCL as a specification language for business rules in database applications. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 104–117. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45441-1_9
4. Djurić, D., Gaševic, D., Devedžic, V.: The tao of modeling spaces. J. Object Technol. **5**, 125–147 (2006)
5. Gaafar, A., Sakr, S.: Towards a framework for mapping between UML/OCL and XML/XQuery. In: Baar, T., Strohmeier, A., Moreira, A., Mellor, S.J. (eds.) UML 2004. LNCS, vol. 3273, pp. 241–259. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30187-5_18
6. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional, Boston (2009)
7. Heidenreich, F., Johannes, J., Seifert, M., Wende, C.: Construct to reconstruct - reverse engineering Java code with JaMoPP. In: Reverse Engineering Models from Software Artifacts - REM 2009 (2009)
8. Henriksson, J., Heidenreich, F., Johannes, J., Zschaler, S., Abmann, U.: Extending grammars and metamodels for reuse: the Reuseware approach. IET Softw. **2**(3), 165–184 (2008)
9. ISO: Financial services - Universal financial industry message scheme - Part 1: Metamodel. ISO 20022–1, International Organization for Standardization (2003)
10. ISO: Information technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-based validation - Schematron. ISO/IEC 19757–3:2016, International Organization for Standardization (2016)
11. Klmek, J., Malý, J., Necaský, M., Holubová, I.: eXolutio: methodology for design and evolution of XML schemas using conceptual modeling. Inform. Lith. Acad. Sci. **26**, 453–472 (2015)

12. Korchagin, A.B., Lisikh, I.G., Nikiforov, D.A., Sivakov, R.L.: Data models for information exchange. Int. J. Open Inf. Technol. **5**(3), 49–55 (2017)
13. Malý, J.: XML Document Adaptation and Integrity Constraints in XML. Ph.D. thesis, Charles University in Prague (2013)
14. Malý, J., Nečaský, M.: Evaluation of OCL expressions over XML data model. J. Univ. Comput. Sci. **20**, 329–365 (2014)
15. Miller, J., Mukerji, J.: MDA guide version 1.0.1 (2003). http://www.omg.org/cgi-bin/doc?omg/03-06-01
16. Moskal, J., Kokar, M., Morgan, J.: Semantic validation of T&E XML data. In: International Telemetering Conference Proceedings, International Foundation for Telemetering, October 2015
17. Nikiforov, D.A.: Development of metamodels using the Eclipse Modeling Framework, September 2015. https://habrahabr.ru/company/cit/blog/266433/
18. Nikiforov, D.A.: An introduction to the development of DSLs using EMFText, November 2015. https://habrahabr.ru/company/cit/blog/270483/
19. Nikiforov, D.A.: The Object Constraint Language, August 2015. https://habrahabr.ru/company/cit/blog/264963/
20. Nikiforov, D.A.: Development of the parser, printer, and editor for SQL using EMFText, December 2016. https://habrahabr.ru/company/cit/blog/271945/
21. Nikiforov, D.A.: Ecore-based metamodels of XML Schema 1.1 and XSLT 2.0, February 2017. https://doi.org/10.5281/zenodo.291483
22. Nikiforov, D.A.: The EMFText-based metamodel, parser, and printer of XPath 2.0, February 2017. https://doi.org/10.5281/zenodo.291481
23. Nikiforov, D.A.: UML to XML Schema 1.1 transformation, version 1.1, February 2017. https://doi.org/10.5281/zenodo.291482
24. Nikiforov, D.A., Korchagin, A.B., Sivakov, R.L.: An ontology-driven approach to electronic document structure design. In: Ignatov, D.I., Khachay, M.Y., Labunets, V.G., Loukachevitch, N., Nikolenko, S.I., Panchenko, A., Savchenko, A.V., Vorontsov, K. (eds.) AIST 2016. CCIS, vol. 661, pp. 3–16. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52920-2_1
25. Nikiforov, D.A., Korj, D.V., Sivakov, R.L.: A survey of tools for XML validation based on the object constraint language (OCL). Inf. Technol. **23**(5), 342–351 (2017)
26. OMG: Object Constraint Language (OCL), version 2.4. Specification, Object Management Group (2014)
27. OMG: UML Profile for National Information Exchange Model (NIEM), version 3.0. Specification, Object Management Group (2015)
28. OMG: Unified Modeling Language (UML), version 2.5. Specification, Object Management Group (2015)
29. OMG: XML Metadata Interchange (XMI), version 2.5.1. Specification, Object Management Group (2015)
30. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation, version 1.3. Specification, Object Management Group (2016)
31. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley Professional, Reading (2009)