Experiences using OCL for Business Rules on Financial Messaging

David Garry
Nomos Software
Rubicon Centre, CIT Campus
Cork, Ireland
+353 21 4928945
david.garry@nomos-software.com

ABSTRACT
In this paper we describe our experiences in using the OCL language in a commercial environment to validate XML-based financial data such as FpML [1] and ISO 20022 [2] data. We describe three problems we have encountered in supporting customer requirements when validating data in this context, and outline how we have chosen to support these requirements. We suggest that it would be useful to define a common approach to solving these problems for users of OCL.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features – *constraints*.

General Terms
Languages.

Keywords
OCL, model driven, case study, experience.

1. INTRODUCTION
The OCL language is a standardized constraint and query language. The language specification [3] is published by the OMG, and is supported in a number of modeling environments.

At Nomos Software, we use OCL in a commercial environment to execute constraints over XML-based financial messaging: in other words we use OCL to implement business rules on financial data. We use our proprietary implementation of OCL.

2. PROBLEMS ENCOUNTERED
When using OCL in commercial environments, we have encountered three very common requirements: easy identification of exact error locations when an OCL constraint fails; support for additional data types and operations (extensions to the OCL standard library); and support for checking against external data sources from within OCL expressions.

2.1 Identifying exact error locations
It is possible, and very useful, to write very general constraints with OCL. However, when a business rule fails, it is important to be able to easily understand why it failed. In our usage scenario, the person investigating the business rule failure only sees the XML data file that is in error, and does not have access to a debugging environment. They need a good description of the problem, as well as the exact data and the location of the data that is in error. For simple constraints, this is straightforward. For general or complex constraints, this can be difficult.

Table 1, for example, shows a constraint on an ISO 20022 'payment status report' message [4] from the European Payments Council Implementation Guidelines [5].

Table 1 ISO20022 OCL Constraint

```
context OriginalGroupInformation20

inv EPC_OrgnlGrp:

StsRsnInf->forAll(a |
  (a.Orgtr.Nm->size() = 1 or
    (a.Orgtr.Id->size() = 1 and
     a.Orgtr.Id.OrgId.BICOrBEI->size() = 1))
   and a.Orgtr.PstlAdr->size() = 0
   and a.Orgtr.CtryOfRes->size() = 0)
```

This constraint states that the originator (Orgtr) of reason information in a payment status report must be identified either by a name (Nm) or by a bank identifier code (BIC). Postal address (PstlAdr) and country of residence (CtryOfRes) cannot be included. This constraint could fail for a number of reasons e.g. because a postal address was included or because a country of residence was included,

In order to to make it possible to return more useful information on the failure reason, we implemented a mechanism to trigger query rules if a constraint fails for a particular context. The OCL author can write one or more query rules and associate them with an OCL constraint; the query rules are triggered if the constraint fails; and the results of the queries are returned to the person troubleshooting the error.

For example, the queries in Table 2 will return information, including line number, on postal address or country of residence fields that were incorrectly included in a payment status report.

Table 2 Query Rules

```
context
OriginalGroupInformation20::getPstlAdr() :
Set(PostalAddress6)
body:  self.Orgtr.PstlAdr->asSet()


context
OriginalGroupInformation20::getCtryOfRes() :
Set(CountryCode)
body:  self.Orgtr.CtryOfRes->asSet()
```

When the constraint in Table 1 executes, these queries are executed against every failed instance of the context.

An alternative approach to solving this problem is outlined in [6]. The authors show that certain patterns of OCL expressions return more useful information for troubleshooting purposes than others. Careful crafting of OCL expressions can help ensure that useful error information is returned at execution time. We note however that, without very good tool support, this makes writing OCL more difficult, and that it may not be possible to provide good debugging information for all scenarios in this way.

2.2 Extending the OCL standard library

The OCL standard library supports 5 primitive types: Integer, Real, Boolean, String, and UnlimitedNatural, each with a number of pre-defined operations [3]. This is very limited. Rules that are straightforward to write in other languages can be difficult to express in OCL. For example, comparing dates is difficult. This is important, e.g., for FpML messaging.

Since we specialise in executing OCL on XML-based models, we chose to support the full set of W3C XML Schema built-in primitive types with additional operations. For example, we support the dateTime type, along with a range of operations such as after, before, allowedDaysInFuture and allowedDaysInPast. This simplifies the expression of dateTime related rules dramatically.

For example, the constraint in Table 3, expressed on the FpML 4-5 model available from [1], checks that a trade's adjusted exercise date is before the adjusted early termination date.

Table 3 – Usage of dateTime operations in OCL

```
context EarlyTerminationEvent
inv ird39:
adjExerciseDate.before(adjEarlyTermDate)
```

Our experience is that support for a broader range of primitive types is useful. Moreover, it is useful to support operations on domain specific types.

Data models for a business domain usually include domain specific types. In ISO20022 messaging, a type is defined for unique bank account identifiers (the IBAN in Europe). Special checks on the IBAN structure must be implemented on the IBAN [1]. To make it easy to write rules on IBANs, we added support for an 'isValidIBAN' operation on the IBAN type. Table 4 shows an example of an OCL expression that invokes this operation.

Table 4 Validating an IBAN in OCL

```
context CashAccount16
inv: PIP_C_93_validIBAN
self.Id.IBAN.isValidIBAN()
```

Making it easy to write OCL for domain-specific data is important. A simple mechanism to extend the OCL standard library for specific domains would be very helpful.

2.3 Referencing external data sources

When executing business rules on messaging data, it is often necessary to compare the data against data in an external database or in a code list defined by some external organisation. For example, fields in payments messaging must comply with code lists maintained by ISO 20022 [7].

To facilitate this, we allow customers to define their own operations on types, and to provide the runtime implementation of the operations. Since the runtime implementations are provided by the customers, they can access customer-specific data sources.

The OCL in Table 5 shows an example of such an operation. The operation isExterrnalFinInstIdCde() is made available on the string type and checks that OrgnlMsgId (original message id) is listed in the financial instrument type external code list.

Table 5 Accessing external data sources with OCL

```
context OriginalGroupInformation20
inv: PIP_C_93_validRef
self.OrgnlMsgId.isExternalFinInstIdCde()
```

A standard mechanism to allow users to be able to invoke external calls from within OCL expressions would be very useful.

3. CONCLUSION

We have outlined three problems encountered when using OCL to execute business rules in a commercial setting. We have provided examples of the problems, and outlined how we resolved them in our applications. We suggest that it would be useful to come up with a common approach to solving these problems for users of OCL.

4. ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

Our thanks to XMLdation [8], for allowing us to use examples from their set of OCL business rules in this paper.

5. REFERENCES

[1] FpML (Financial products Markup Language) http://www.fpml.org/

[2] ISO 20022 (ISO Standard for Financial Services Messaging) http://www.iso20022.org/

[3] Object Constraint Language, Version 2.3.1. OMG Document Number : formal/2012-01-01. http://www.omg.org/spec/OCL/2.3.1/.

[4] Payment status report message, pain.002.001.03.xsd, http://www.iso20022.org/message_archive.page

[5] SEPA Credit Transfer Scheme Customer-to-Bank Implementation Guidelines Version 6.0, http://www.europeanpaymentscouncil.eu/knowledge_bank_detail.cfm?documents_id=537

[6] Chiorean, D., Vladiela, P., and Ober, I. 2012. Testing-oriented improvements of OCL specification patterns. In *Proceedings of AQTR 2010* (IEEE, volume 2, pages: 1-6)

[7] ISO20022 External code list definition. http://www.iso20022.org/external_code_list.page

[8] XMLdation http://www.xmldation.com/

[9] IBAN structure http://www.tbg5-finance.org/?ibandocs.shtml