Auto-generation of Model Visitor Frameworks

Adolfo Sánchez-Barbudo Herrera

Department of Computer Science, University of York, UK.
asbh500@york.ac.uk

Abstract. The visitor pattern, a well known Gang of Four design pattern, provides a suitable way to add operational behaviour to models. However, as soon as the number of metamodels and visitor implementations start to grow, some of the pattern shortcomings make its usage less convenient. This paper presents how the synergy between the Visitor pattern and MDE has been addressed by two open source projects: Eclipse OCL and QVTd. As a result, a visitors framework generator is proposed to alleviate some of the visitor pattern shortcomings.

1 Challenge

The visitor pattern is one of the behavioural patterns described in the Gang of Four software design patterns [1]. The flexibility that this pattern provides, by the means of allowing the decoupled addition of behaviour to a collection of objects without any need to change their corresponding classes, makes this pattern attractive in object oriented software design and related research [2]. In Model-Driven Engineering (MDE), research on visitors has received less attention, with a few known works [3, 4].

Whereas applying the visitor pattern to models might be considered straightforward in the Eclipse Modeling Framework (EMF) [5], it turns to be a task that is, at least, as tedious as performing it in a traditional programming language. For example, every X EClass requires an *accept* EOperation; a visitor EClass requires a *visitX* EOperation for every X EClass. If the number of involved metamodels (and subsequently metaclasses) grows large, the shortcomings associated to the visitor pattern hinder its adoption.

Provided the potential benefits that the visitor pattern can bring to models, this paper focuses on the challenge presented by the limitations of using the visitor pattern in MDE, which we propose to overcome by reducing the amount of manual intervention required to realize the visitor pattern. Specifically, this challenge has been tackled in the context of the Eclipse OCL[1] and Eclipse QVTd[2] projects by the means of a visitors framework generator.

[1] https://projects.eclipse.org/projects/modeling.mdt.ocl
[2] https://projects.eclipse.org/projects/modeling.mmt.qvtd

2 Why a Visitors Framework Generator in MDE?

Eclipse OCL and Eclipse QVTd are two projects conceived to support MDE by providing model management languages and tools, in which a substantial number of EMF based metamodels are involved. The reasons of introducing model visitors come from the original visitor pattern: the operational behaviour of a model is centralised in a visitor class; and the metamodel doesn't need to be changed every time a new behaviour is needed, which is convenient to third party consumers who can not alter the metamodel at all.

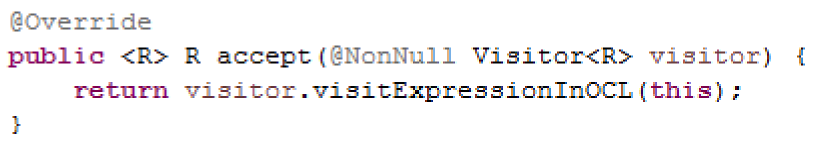
However, given the substantial number of metamodels and comprised metaclasses, introducing the pattern is rather a tedious task. Additionally, evolving the visitors along with any change to the underlying metamodel is troublesome, as identified in the original design pattern [1]. To alleviate this situation, MDE techniques can be adopted. In particular, automated code generation can be used to produce visitor frameworks from metamodel definitions. On top of the framework, actual visitor implementations can provide particular operational behaviours (e.g. evaluators, pretty printers, etc.).

With the proposed generator we not only leverage the barrier of model visitors creation, but also we can alleviate some of the shortcomings of the visitor pattern in some metamodel evolution scenarios [4]: for example, by having an automatically generated framework of visitors for a specific metamodel, we could keep existing visitor implementations working when a new metaclass is added, by just regenerating the model specific visitors framework.

3 Visitors Framework Generator in Eclipse OCL/QVTd

In Eclipse OCL there is a MDE-based tool which facilitates the generation of metamodel-specific visitors framework. The main features to highlight:

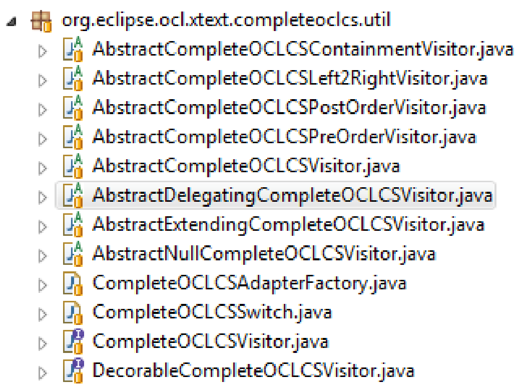
High degree of automation: For a given metamodel, the visitor pattern is weaved into the generated metamodel implementation, so that for every non-abstract *X* class an accept method is generated, and whose implementation will delegate to the corresponding *visitX* method of a *Visitor* interface, as depicted by Figure 1.



```
@Override
public <R> R accept(@NonNull Visitor<R> visitor) {
    return visitor.visitExpressionInOCL(this);
}
```

Fig. 1. Generated accept method for an *ExpressionInOCL* class

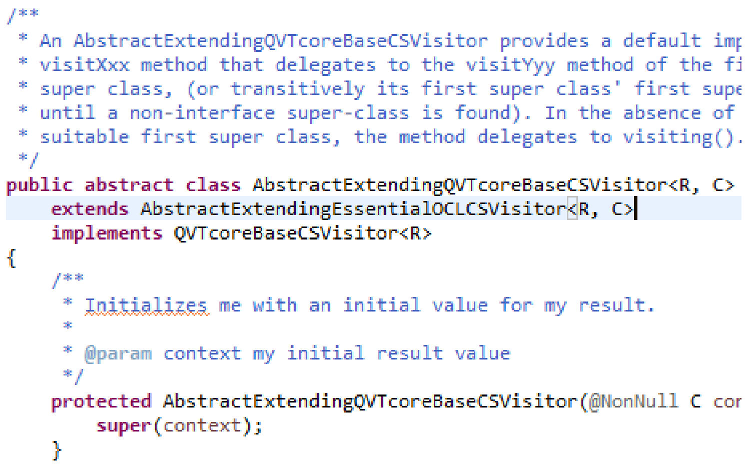
Additionally, a framework of abstract visitors with various purposes[3] is also generated. Figure 2 gives a brief overview of the framework for a particular OCL metamodel.



```
org.eclipse.ocl.xtext.completeoclcs.util
    AbstractCompleteOCLCSContainmentVisitor.java
    AbstractCompleteOCLCSLeft2RightVisitor.java
    AbstractCompleteOCLCSPostOrderVisitor.java
    AbstractCompleteOCLCSPreOrderVisitor.java
    AbstractCompleteOCLCSVisitor.java
    AbstractDelegatingCompleteOCLCSVisitor.java
    AbstractExtendingCompleteOCLCSVisitor.java
    AbstractNullCompleteOCLCSVisitor.java
    CompleteOCLCSAdapterFactory.java
    CompleteOCLCSSwitch.java
    CompleteOCLCSVisitor.java
    DecorableCompleteOCLCSVisitor.java
```

Fig. 2. Generated visitors framework for a particular OCL meta-model

Support for derived metamodels: QVT languages reuse OCL as their expression languages. Therefore, the tooling provides generation of visitors which can extend the visitors generated for a different metamodel they might extend and/or use. Figure 3 shows a snippet of a particular QVTd visitor of the generated visitors framework.



```
/**
 * An AbstractExtendingQVTcoreBaseCSVisitor provides a default imp
 * visitXxx method that delegates to the visitYyy method of the fi
 * super class, (or transitively its first super class' first supe
 * until a non-interface super-class is found). In the absence of
 * suitable first super class, the method delegates to visiting().
 */
public abstract class AbstractExtendingQVTcoreBaseCSVisitor<R, C>
    extends AbstractExtendingEssentialOCLCSVisitor<R, C>
    implements QVTcoreBaseCSVisitor<R>
{
    /**
     * Initializes me with an initial value for my result.
     *
     * @param context my initial result value
     */
    protected AbstractExtendingQVTcoreBaseCSVisitor(@NonNull C cor
        super(context);
    }
```

Fig. 3. Excerpt of a derived visitor

[3] Getting into details of the framework is beyond the scope of this paper

4 Visitors Framework Generator Implementation

As the reader might understand, we can't get into details in this short paper about the generated visitors framework or provide implementation details about the own generator. However, in this section we give a brief overview of the developed tooling as well as mentioning some of the involved third party tools.

Figure 4 depicts the overall approach. The input ❶ to the generator is an EMF [5] GenModel filer referring to an Ecore file that corresponds to the target meta-model. The GenModel contains additional annotation to provide information relevant to the generator (e.g. fully qualified name for the *Visitor* and *Visitable* interfaces). The generator ❷ is based on the Model Workflow Engine (MWE) [6] technology, and the developed MWE components are responsible for:

- Generating a *'visitable'* model java implementation ❸. This involves invoking the EMF generator, which will generate the enhanced implementation by inserting the corresponding *accept* methods for every model class. We developed EMF JET templates to achieve this insertion.
- Generating the model-specific visitors framework ❹. This includes generating the *Visitor* and *Visitable* interfaces and a set of default abstract visitors implementation. Giving implementation details about these default visitors goes beyond the goal of this paper.

With all of this, creating specific behaviours for the model can be achieved by implementing manual visitors ❺ that extend the appropriate default visitor of the generated framework.

Alternatively, by the means of additional MDE-based tools ❻, we could also generate more specific visitors ❼ that comprise a particular modelled behaviour.

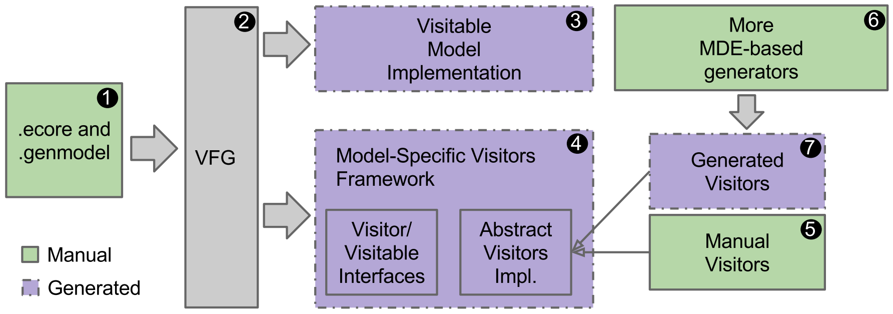

❶
.ecore and .genmodel
❷
VFG
❸
Visitable Model Implementation
❹
Model-Specific Visitors Framework
Visitor/ Visitable Interfaces
Abstract Visitors Impl.
❻
More MDE-based generators
❼
Generated Visitors
❺
Manual Visitors
Manual
Generated

Fig. 4. Overall overview of the visitors framework generator

5 Visitors usage in Eclipse OCL/QVTd

To conclude, we provide an overview about how the visitors framework generator is extensively used within Eclipse OCL and QVTd projects. Table 1 provides detailed measurements about the generated visitors usage. The table variables are defined as follows:

– V_n: Denotes the number of involved metamodels on which the visitor pattern has been applied.
– V_a: Denotes the number of visitors classes which are automatically generated as part of the visitors framework[4].
– V_m: Denotes the number of visitors classes which are manually implemented, and which rely on the automatically generated visitors framework.

Project	V_n	V_a	V_m
OCL	7	48	67
QVTd	10	89	76

Table 1. Measurements of visitors usage in Eclipse OCL & QVTd

Given the high amount of visitor implementations, and the substantial auto-generated infrastructure that the model specific visitors frameworks provide, it can be concluded that both the visitor pattern and the framework generator have turned to be fundamental within the Eclipse OCL and QVTd projects.

Acknowledgement. I gratefully acknowledge the support of the Engineering and Physical Sciences Research Council (UK) via the LSCITS initiative.

References

1. Erich Gamma, Richard Helm, John Vlissides, and Ralph Johnson. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1994.
2. Tanumoy Pati and James H Hill. A survey report of enhancements to the visitor software design pattern. *Software: Practice and Experience*, 44(6):699–733, 2014.
3. Holger Krahn, Bernhard Rumpe, and Steven Völkel. Monticore: a framework for compositional development of domain specific languages. *International journal on software tools for technology transfer*, 12(5):353–372, 2010.
4. Adolfo Sanchez-Barbudo Herrera, Edward D. Willink, Richard F. Paige, Louis M. Rose, and Dimitrios S. Kolovos. Automatic application of visitors to evolving domain-specific languages. In Sam Simpson, editor, *Sixth York Doctoral Symposium*, pages 46–54. University of York, October 2013.
5. Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2008.
6. Modeling Workflow Engine 2 Documentation. On-Line: `https://eclipse.org/Xtext/documentation/306_mwe2.html`.

[4] Interfaces such as Visitor/Visitable are excluded