Electronic Communications of the EASST
Volume 36 (2010)

Proceedings of the Workshop on OCL and Textual Modelling (OCL 2010)

A Feature Model for an IDE4OCL

Joanna Chimiak–Opoka, Birgit Demuth

15 pages

Guest Editors: Jordi Cabot, Tony Clark, Manuel Clavel, Martin Gogolla
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Feature Model for an IDE4OCL

Joanna Chimiak–Opoka[1], **Birgit Demuth**[2]

[1] University of Innsbruck, Austria, joanna.opoka@uibk.ac.at
[2] Technische Universität Dresden, Germany, birgit.demuth@tu-dresden.de

Abstract: An Integrated OCL Development Environment (IDE4OCL) can significantly improve the pragmatics and practice of OCL. Therefore we started a comprehensive requirement analysis with the long term vision of a multisite IDE4OCL project. In this paper we present a feature model for the IDE4OCL vision based on this analysis. In an earlier work we identified domain concepts, tool–level interactions with IDE4OCL, and use cases for OCL developers including a set predefined features. In the second step, we asked the OCL community members for their feedback on our proposal. Around 100 researchers, tool developers and practitioners who gained experience with OCL have voted in an online–survey. The results gave us a valuable insight in the needs of OCL usage both in usual and advanced OCL applications. One of the important results is a collection of features that have been proposed additionally to our predefined features. We analysed all the comments of the participants of the survey and consolidated them into an extended set of IDE4OCL features and eventually into a feature model.

Keywords: OCL, modeling, feature model, integrated development environment

1 Introduction

Thirteen years ago, the first version of the Object Constraint Language (OCL) was published as a part of the Unified Modeling Language (UML) standard. Since the publication of the OMG OCL standard many academic and commercial OCL tools have been developed such as Dresden OCL[1], USE[2] and MDT OCL[3]. In recent years OCL support is also implemented in modeling tools as in Magic Draw UML[4], Borland Together[5] and ECO for Visual Studio[6]. An overview of further OCL tools can be found in the OCL Portal[7]. This seems to affirm the following rule of thumb [Rid84]: *It is commonly thought that 10 years is needed for technology to pass from its initial conception into wide–spread use*. Below we will focus on two aspects important for a wide–spread use; namely maturity and usability of OCL, related to standards and tools.

The question of the **maturity** of modeling, in particular modeling with UML, was discussed during a MODELS 2009 keynote [Mel09]. Obviously OCL is less mature than UML, but the

[1] http://dresden-ocl.sourceforge.net/
[2] http://www.db.informatik.uni-bremen.de/projects/USE/
[3] http://wiki.eclipse.org/MDT-OCL/
[4] http://www.magicdraw.com/
[5] http://www.borland.com/us/products/together/
[6] http://www.capableobjects.com/ProductsServices_ECO.aspx
[7] http://st.inf.tu-dresden.de/oclportal/

maturity of OCL has been improved in the recent years. Considering the amount of research work related to OCL and the number of academic and industrial OCL tools [CGG08], one can see a huge potential for OCL to be in wide–spread use in the future. However, changes and imperfections of OCL specification constitute a crucial obstacle in achieving full maturity of the language.

Another important issue related to the adoption of a technology is its **usability**. As indicated in [Con09] software users and their usability issues seem to be commonly neglected in literature and practice. The availability of multiple OCL tools has caused the growth of the OCL community in the academic and industrial context. However, the OCL community has not agreed, until now, what functionality is important for OCL users. The topic of the last OCL Workshop [CCG+09] was therefore pragmatics of OCL. The question of usability of an OCL tool was already discussed in [CPP08], where it was mentioned that *tools' constituents (editors, compilers, browsers) must implement the functionalities established by integrated development environments* (IDEs).

We went one step further and published a systematic requirements analysis for such an **integrated development environment for OCL** (IDE4OCL) [CDSR]. We identified domain concepts, tool–level interactions with IDE4OCL, and use cases for OCL developers including a set of 21 predefined features. To identify use cases and features we analysed our personal experience with academic and industrial users of OCL and lessons learned from the well–established domain of IDEs for programming languages. A comprehensive description of the identified use cases and features, discussion of architectural decisions and technical feasibility can be found in our prior paper.

To validate our ideas, in the second step, we asked members of OCL community for their feedback. In this paper we present **qualitative**[8] **results of our on–line survey**, in which around 100 researchers, tool developers and practitioners, who gained experience with OCL, expressed their opinions. The results gave us a valuable insight in the needs of OCL usage both in usual and advanced OCL applications. One of the important results is a collection of features that have been proposed in addition to features predefined by us. We analysed all comments of the participants of the survey and consolidated them into an extended set of IDE4OCL features eventually presented as a feature model.

The feedback from the OCL community is very important to us, as our long term goal is a **community driven development** of an IDE4OCL with partners from the academic and industrial context. Moreover, we aim to network OCL users[9]. According to our knowledge, our research is the first attempt at questioning the global OCL community to systematically gather requirements to develop an IDE for OCL. Our goal is to gain a common understanding of the important functionality for OCL–driven modeling and development.

The rest of the paper is structured as follows: in Section 2 we outline the idea of an IDE4OCL. In Section 3 we shortly describe our survey–based research and give respondents characteristics. The collections of features proposed by us and by respondents of our survey are included in Section 4. Based on both collections, features are summarised in Section 5 and presented as a feature model for an IDE4OCL. Finally, in Section 6, we summarize our results and indicate future work.

[8] The quantitative evaluation of the results is a topic of a future paper.
[9] LinkedIn group of OCL users at http://www.linkedin.com/groups?gid=3007822

2 IDE4OCL Overview

In this section we summarize the idea of IDE4OCL presenting its context, use cases and features.

In recent years, many OCL based tools and tool–chains have been developed for multiple purposes. Currently, the OCL landscape is rich and heterogeneous. For the purpose of the context specification of IDE4OCL we propose a simplified view of an **OCL tool landscape** with tools directly interacting with IDE4OCL (Fig. 1). The view is based on possible usage scenarios of OCL discussed in [DW09]. The landscape specification also shows what IDE4OCL is not intended to be (components that are other ones than in the diagram) and at the same time points out, which functionality is considered to be outsourced (realisations in the other components in the diagram).

The **architectural style** of IDE4OCL can be considered as a toolset [HDF02] or a collection of plug–ins. We assume the possibility of OCL expression exchange between tools with full preservation of their semantics.

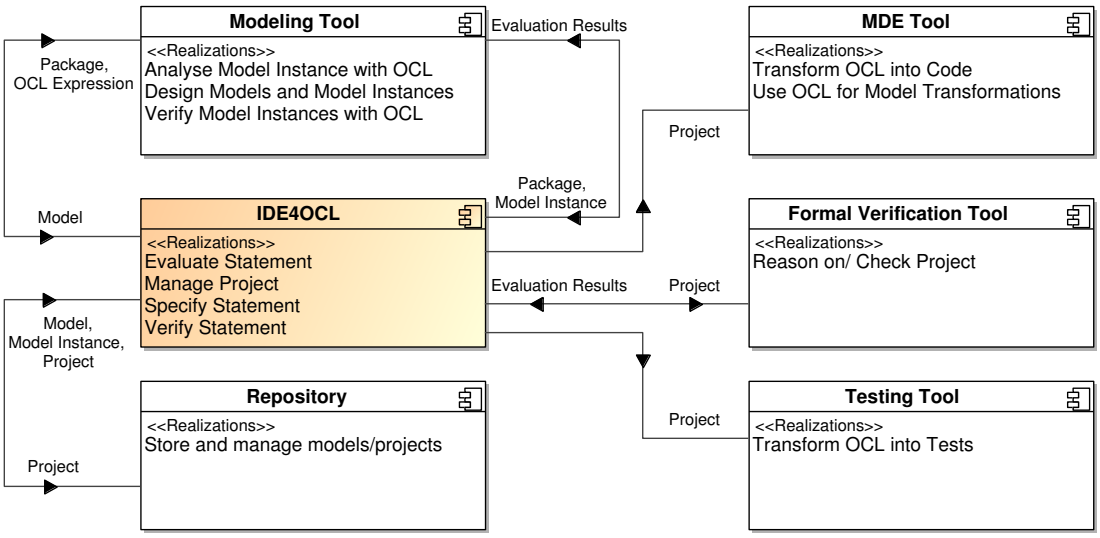


Figure 1: The OCL tools landscape: relations between tools (from [CDSR]).

From the *functional point of view*, we distinguished four main use cases to be realised by the IDE4OCL component (Fig. 1), namely specification, evaluation and verification of statements and project management.

Specify Statement is the basic use case of an IDE4OCL, where an OCL developer specifies an OCL statement[10]. Note that the OCL statement can be specified both on the metamodel and the model layer. We consider here the creation of a new statement or a package from scratch or the modification of an existing one.

[10] The term *OCL statement* was introduced in [CDSR] for pragmatic reasons to denote a single artefact of an OCL specification that can be developed within an IDE4OCL.

Evaluate Statement is a use case where statements are parsed by an OCL parser that creates an abstract syntax model that is evaluated by an OCL interpreter. This executes the statements defined on the model for the model instance. This use case can be performed on request from an OCL developer or from another tool in the OCL tool landscape (Fig. 1). As mentioned in [CDSR], we consider the evaluation in the form of code generation as outsourced functionality.

Verify Statement use case covers both formal and empirical approaches to verification and validation of an OCL specification. Due to the complexity of formal verification, we consider it to be outsourced. As a form of empirical validation of statement, IDE4OCL should support testing of OCL statements (OCLUnit, [CO09]).

Manage Project use case is required for efficient support of OCL development. In the case of large projects it is important to have efficient management of all artefacts and configuration features of the IDE4OCL itself. This use case covers all management issues within IDE4OCL and related communication with other tools.

From the *implementation point of view*, the Specify and Evaluate Statement use cases may be hardly distinguishable. Modules to edit, to parse and to interpret OCL statements are crucial parts of IDE4OCL. For high usability, a user–friendly editor is indispensable and must fulfill the typical features of the Specify Statement use case. Nowadays, modern parsers go beyond pure parsing and are powerful tools offering advanced editing support. An OCL parser has also to semantically check OCL statements against the underlying model. A parser and editor is often shortly referred as an *OCL editor*.

For the Specify Statement use case we identified 21 features (among others Auto–completion, Basic Editing, Debugging and Syntax Highlighting), which were used in the on–line survey described in the next section. The features themselves are described in Section 4.2.

3 Survey–Based Research

In this section we describe the design of the survey (Section 3.1), data collection issues (Section 3.2), and respondents characteristics (Sections 3.3). Features proposed by respondents are described in Section 4.2.

3.1 Survey Design and Structure

The first step in our empirical research [BW84] was the design of the survey. The **goal of the survey** was to evaluate the appropriateness and completeness of the set of predefined features. The survey consists of four parts: personal data, appropriateness of features, completeness of the features and feedback. As the goal of this paper is to provide analysis of the qualitative survey results, we focus mostly on feedback related to achieve completeness of the futures.

Personal data part was designed to collect information about respondents. The results are presented in Section 3.3.

Appropriateness of features part dealt with importance and urgency of the predefined features. The results of the statistical analysis will be published in an future paper. Here we use statistical evaluation results to select mandatory features for the feature model in Section 5.

Completeness of the features part was to ask the respondents on their opinion about the completeness of the set of the predefined features and to propose extensions for each use case. All but the first question in this part were open text questions, thus the respondents were free to write any information there. The summary of this part is given in Section 4.2 and Section 5.

Feedback part consisted of the inquiry of respondents' feedback in form of free text comments. These so–called *open questions* are spread over survey sections according to their content.

3.2 Data Collection

The second step of our research, data collection, was a difficult task, as there is no easy way to identify OCL users. To collect data we used different channels: personal and electronic. We used hard copies and an on–line survey engine[11].

Initially, data was collected through **personal contact** with respondents. Our first opportunity to encounter the OCL community was at the MODELS 2009 conference, especially at the OCL Workshop. We distributed our survey after the presentation and discussion of the idea of IDE4OCL [CDSR]. The second opportunity to meet people related to OCL was the 10th anniversary of Dresden OCL[12] where we distributed our survey to the participants. Additionally, we contacted our co–workers and students in Dresden and Innsbruck.

After the phase with direct contacts, we started to use the **electronic channels**: mailing lists, portals, e–mails and invitations via the survey engine with pre–generated tokens. We identified and directly contacted over 200 people using OCL in the academic and industrial context[13] and collected data over a period of 12 months[14].

3.3 Respondents Characteristics

In the personal data part of the survey we asked respondents to provide information on their experience with OCL and the context in which they are using it. Additionally we asked for self–estimation of their OCL knowledge. Below we shortly describe these dimensions and present corresponding diagrams (Fig. 2).

The **distribution of experience** based on 5 multiple choice kinds of experience is shown on the left side of Fig. 2. The largest group consists of users (E5), the next groups of tool developers (E3) and researchers (E4). Several respondents were involved in standardisation processes (E1, E2).

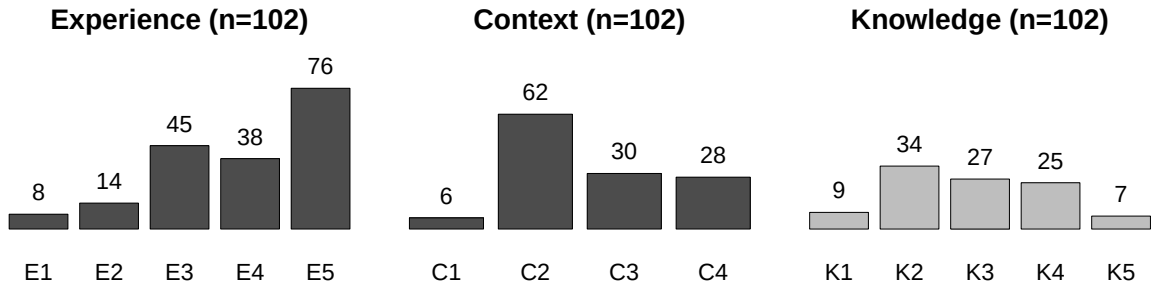
The **distribution of contexts** based on 4 multiple choice options of occupation is shown in the middle of Fig. 2. The largest group of respondents came from research institutions (C2), the next groups are employees of companies (C3) and students (C4). Only several respondents are working for standardisation organisations (C1). The greater amount of respondents from

[11] http://www.limesurvey.org/

[12] http://dresden-ocl.sourceforge.net/10years.html

[13] This number is based on the number of invitations from the survey engine and excludes people contacted indirectly by broadcasting (via portals, mailing lists and e–mails). Considering the direct contact only, the response rate was 48%.

[14] The survey is still open and available at http://squam.info/ide4ocl/.
We will monitor and analyse its results from time to time.



E1: OCL standardisation
E2: OMG standardisation
E3: OCL tool development
E4: OCL research
E5: OCL usage in a modeling/development process

C1: work for OMG
C2: work in a research institution
C3: work in a commercial company
C4: being a student

K1: novice user
K2: user
K3: advanced user
K4: expert
K5: guru

Figure 2: Distribution of different experience, context (multiple choice options, in dark grey) and knowledge types (single choice options, in light grey).

research institutions compared to companies may be interpreted as a sign of wider usage of OCL in the academic rather than the industrial context.

This observation and its interpretation can be caused by the academic bias, which makes generalisation vulnerable. It is easier to contact respondents from the academic context, i.e. students and researchers, because research on OCL is published. In contrast, information on usage of OCL in the commercial context is harder to find. Furthermore, the authors in social networks are predominantly settled in the academic context.

The **distribution of knowledge** based on 5 disjoint self–estimation levels is presented on the right side of Fig. 2. Central groups (K2–K4) are more dominant than peripheral ones (K1, K5).

4 Collection of Features

In this section we present features, which covers features defined in a solid requirements analysis by the authors of the survey (Section 4.1) and further features proposed by the respondents of the survey (Section 4.2).

4.1 Predefined Features

In the following, we present shortened definitions of 21 predefined features. For complete definitions we refer to [CDSR] or to the web[15].

Association End Navigability should be supported independently of the navigability of the underlying association in the model.

[15] http://squam.info/?p=325

Auto–completion enables predicting a word or phrase that the user wants to type in without the user actually typing it in completely. Not only the OCL grammar but also the underlying model (model context, attributes, operations, ...) has to be included in the auto–completion mechanism.

Auto–indentation helps to better convey the structure of code to human readers (e.g. to show the relationship between OCL nested structures).

Basic Editing is a set of features related to editing any kind of text documents, which can be useful when editing OCL statements, e.g. spell checking or regular expression based find and replace.

Code Folding enables the user to selectively hide and display sections of an edited file.

Collaborative Editing allows several people to edit a file using different computers and provides a possibility of team/pair work to enable knowledge transfer (e.g. teacher and student, geographically distributed developers).

Debugging should support developers in understanding the nature of bugs and typically offers functions such as running an expression step by step, conditional breakpoints in an expression to examine the current state, tracking and changing the values of variables. Additionally, logging of and test generation based on debugging activities may be supported.

Document Interface is a set of features supporting editing of multiple documents. It covers support of multiple instances, single and multiple document window splitting, multiple document overlappable windows and a tabbed document interface. While working with hybrid models, it enables traceing relationships between textual and graphical notations.

Hybrid OCL/MOF View should provide an abstract syntax model and additionally highlight the context of any OCL expression in the metamodel.

Macro Mechanism enables short sequences of keystrokes and mouse actions to be transformed into other, usually more time–consuming, sequences.

Name Resolution for model elements in the form of simple or package–qualified names is required.

Profiler enables performance analysis based on OCL statement/package evaluation time. It can be used to determine which OCL statements are most frequently evaluated and focuses on their optimization.

Refactoring Support is useful for renaming and restructuring entities whilst preserving their original semantics. For full support, dependencies between statements must be analysed to perform a series of renaming activities. Also, extracting a definition or a template from a statement should be supported to avoid code duplications.

Reuse Support can be realized at different levels. At the same abstraction level, OCL code can be reused by the composition of statements, template and library import mechanisms. A specification from the higher abstraction level can be reused during development of a specification at a lower level to ensure the correctness of metamodel instantiation.

Statement Element Browser provides facilities to browse, navigate, or visualize (e.g. as an outline) the structure of an OCL project, including OCL statements, elements and element instances.

Statement Coverage is used to measure, based on coverage criteria, the degree to which an OCL specification has been tested.

Static Statement/Specification Analysis is the analysis conducted without evaluation of a state-

ment/specification and allows implicit typing or displaying size and complexity metrics.

Symbol Database enables quick and easy location of statements, elements, element instances and so on based on indexing.

Syntax Highlighting enables the display of OCL code in different colors and fonts according to the category of terminal symbols. Additionally highlighting for an underlying metamodel, errors and brace matching should be considered.

Template Support enables the definition, usage and management of templates. It is related to refactoring and reuse features.

Visibility and Lexical Scoping MOF provides the context–sensitive interpretation of a package during the name resolution of OCL constraints in the context of a package with two distinct interpretations (depending on its role as a source or as a target of a package merge or import relationship).

The features were intended for the Specify Statement use case, but they partially cover other use cases. In Fig. 3 we present **coverage of use cases** by the set of predefined features.

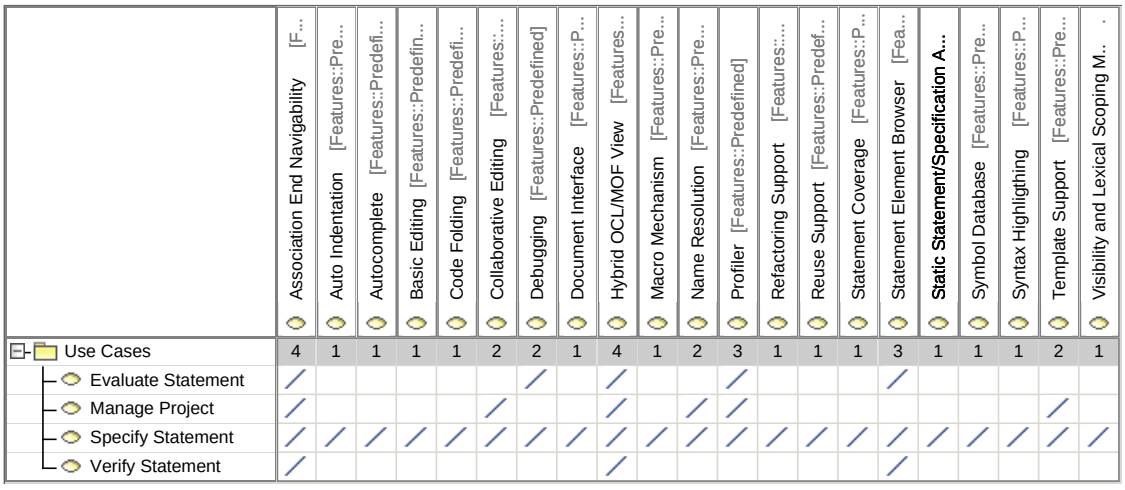


Figure 3: Use cases and their related predefined features

4.2 Proposed Features

In the question on completeness on predefined features for the Specify Statement use case 68% of respondents answered positively. 59 respondents left their qualitative feedback and among them 46 answered questions on proposed features for the specify (23), evaluate (31), verify statement (27) and manage project (27) use case. We analysed and grouped their proposals and selected 13 most frequently mentioned features. Below we provide definitions of the proposed features and next we show their relation to the use cases (Fig. 4).

Batch Mode is a usage of IDE4OCL without activating its GUI. It enables running IDE4OCL as a back–end tool or incorporating it into a tool chain. A full and partial batch mode

can be considered depending on the user interaction required. Input for the batch mode can be provided from a command line, scripting mechanism or via configuration files. An example usage scenario is the evaluation of model metrics defined in OCL and passing their values to another tool or storage. Another example can be the verification of an OCL project by testing all OCL expressions (running OCLUnit tests [CO09]) before committing an OCL project into a version control system.

Documentation Specification and Generation provides an in–code documentation mechanism that is popular for many programing languages, e.g. Javadoc[16]. This mechanism enables keeping documentation together with code and generate documents based on the current state of the development [HT99]. We proposed adaptation of this practice to the OCL context [CO09].

Error Handling is generally the detection and resolution of errors and warnings at different levels. In the OCL context error handling refers both to the static and the dynamic semantics of OCL expressions. The user's requirements for an efficient development of OCL statements include instant error/warnings detection while editing OCL, detection of all errors/warnings in an OCL package at a time (i.e. error reporting should not stop after the first error). as well as immediate, helpful and all in all user adequate visualization of errors/warnings respectively messages. Beyond error detection, developers wish support for quick fixes in OCL specification and evaluation.

Interfaces to Other Tools enable cooperation between IDE4OCL and other tools in the OCL landscape. We decided to outsource some issues to avoid extensive complexity and heterogeneity. The interfaces should enable passing selected OCL statements to these tools to link the outsourced functionality. If appropriate, they should display feedback from them in the IDE4OCL. To integrate particular tools, adaptors have to be created.

OCL and Model Perspective provides two perspectives: one with OCL as main artifact and one with models in focus. In some cases, especially in the specify statement use case, it is easier to work within the OCL perspective. For example, when refactoring, reusing or optimising OCL expressions. In other cases, especially in the evaluate statement use case, it is more comfortable to work within the model perspective. For example, when evaluating OCL expressions results may be visualised in a corresponding model/diagram. Model elements can be enriched with annotations based on OCL evaluations results. It increases comprehensiveness of models and gives better insight into their violations of OCL–specified quality restrictions.

OCL Compliance is a prerequisite for realizing interoperability with other tools, which is one of the important architectural issues of an IDE4OCL. The OCL specification[17] distinguishes in its Chapter 2 three kinds of OCL compliance: syntax, XMI and evaluation compliance. OCL 2.0 was developed in parallel with UML 2.0 and MOF 2.0 that share a common core, thus the OCL standard requires that each OCL tool has to declare what compliance points against UML 2.0 and MOF 2.0 are fulfilled. To the best of our knowledge we know no OCL tool that describes its compliance. In many cases OCL tool support

[16] http://java.sun.com/j2se/javadoc/—a tool for generating API documentation in HTML format from doc comments in Java source code.

[17] OMG OCL2.2: http://www.omg.org/spec/OCL/2.2/

is restricted to class diagrams, but in practical applications support of other diagrams is important, e.g. of state machines, like in ECO for Visual Studio.

OCL Testing is a supportive mechanism for OCL developers. It enables to check if a defined OCL statement evaluation provides expected results based on a test model. However, *testing shows the presence, not the absence of bugs*[DBR69], it is the best practice to increase the quality of code. We proposed and discussed unit testing of OCL in [CO09].

Scalability is an often required feature enabling evaluation on large models, large collections of objects, or large OCL packages, but it has not yet been investigated to a sufficient extent. Support of configurations or incremental approaches can be used as a realisation approach of this feature. In the support for configurations, constraints in particular invariants should be evaluated in relevant situations to reduce the effort for useless or costly evaluations such as in the case of the OCL22Java generator[18]. In incremental integrity checking, only the subset of the system state that has been changed is considered [CT09].

Support of Libraries enables to increase modularisation and reuse of OCL statement as it provides import mechanism. It can be used to provide libraries of typical statements for the OCL standard library either as a learning material or for reuse purposes. Additionally, for user convenience, on–fly libraries could created, e.g. for all OCL statements relating to a particular context to be editable together. Moreover, it should be possible to bind libraries in another formats or in another storage, e.g. as jar files or from databases. An example scenario: A UML profile comes with the constraints defined. These constraints could be compiled into the library, and then the tool, using this profile can just access/run constraint implementations from the (jar) library.

Support of OCL Modularisation can be interpreted from various perspectives. One is a technique for the support of user–defined modularisation of the language itself. This is useful to extend, adapt or embed OCL expressions for different usage[19] as well as to reduce the complexity of OCL [AZH08], [WTZ10]. Another OCL modularisation feature is the modularisation of OCL packages. For example, this could be useful for reuse of OCL expressions in the case of collaboration of partners who share the same model but not all constraints. A technique that we recommend for it is the usage of libraries.

Syntax and Semantics Extensions can increase acceptance of OCL by modifications of its syntax that resembles a formal language. Therefore the support for creation of an alternative concrete syntax is required. Another idea is to introduce notational beautifiers that means a printer that exports OCL programs using a math–inspired short–hand notation [Sue06]. Extending the semantics of OCL besides the syntax is much more difficult but also desired. Examples are the introduction of regular expressions or ad–hoc polymorphism. It should be indicated that OCL language modularization is a technique implementing syntax and semantics extensions.

Variability of Metamodels and Technical Spaces is required to support the original intention of OCL, i.e. to specify constraints on UML models and well–formedness rules in metamodeling. Along the model–driven software development and domain specification lan-

[18] http://dresden-ocl.svn.sourceforge.net/viewvc/dresden-ocl/trunk/ocl20forEclipse/doc/pdf/manual.pdf

[19] see presentation of Christian Wende et al at the Workshop Dresden OCL - Quo Vadis at http://dresden-ocl.sourceforge.net/10years.html

guages hype, OCL is very often used on various metamodels. Consequently, parsers and interpreters are needed that support variability of metamodels. Regarding interpreters, the used metamodel is not the only issue. In practice, OCL interpreters are implemented in a specific technical space such as Java, EMF or a proprietary model repository. To be generic against models in variable technical spaces, an IDE4OCL should provide a framework to evaluate OCL expressions on objects in different technical spaces [WTW10].

Version Management is desired to have version control for both OCL expressions and models. It should provide support for change management and change tracking. Versioning can help to integrate OCL statements into the software life cycle. A minimalistic solution is an integration with subversion or CVS. However, a target solution should be more mature and dedicated for versioning of OCL and models at statement and model element level.

	Batch Mode [Features::Pro...	Documentation Specificati...	Error Handling [Features::...	Interfaces to other Tools [...	OCL and Model Perspectiv. ..	OCL Compliance [Feature...	OCL Testing [Features::Pr...	Scalability [Features::Prop...	Support of Libraries [Feat...	Support of OCL Modularis...	Syntax and Semantics Ext...	Variability of Metamodels ...	Version Management [Fea...
Use Cases	2	2	3	4	4	4	4	4	4	4	4	4	1
Evaluate Statement	/		/	/	/	/	/	/	/	/	/	/	
Manage Project		/		/	/	/	/	/	/	/	/	/	/
Specify Statement		/	/	/	/	/	/	/	/	/	/	/	
Verify Statement	/		/	/	/	/	/	/	/	/	/	/	

Figure 4: Use cases and their related proposed features

5 An IDE4OCL Feature Model

In the previous section we presented sets of predefined (Section 4.1) and proposed features (Section 4.2) without any prioritisation of them. In this section we structure features into a comprehensive IDE4OCL feature model, where we distinguish between mandatory and optional features and group them into more general categories.

The feature model should give a better understanding and overview of the features for both OCL users and OCL tools developers. For researchers on the OCL related topics, it can serve as an overview of topics crucial to OCL users (categories of features). Moreover, it can serve as a basis for development of customised IDE4OCLs in the software product line approach and as an aggregation and visualisation framework in tools comparison.

For the predefined features we used quantitative evaluation of the survey (as described below) and due to the lack of such evaluation for proposed features we based our classification on personal experience with OCL and IDEs. Below we describe our strategy of feature classification in the feature model depicted in Figure 5.

- The predefined features with the statistically highest importance and at the same time highest urgency ranking in the survey [20] are declared as *mandatory features*: Auto–completion, Basic Editing, Syntax Highlighting, Debugging and Refactoring Support. The first three features can be corresponding to the state-of-the-art considered as basic mandatory features. Debugging and Refactoring Support are in contrast features that are not yet implemented in OCL tools. Therefore these both features are a challenge for OCL tool developers in the near future. All other predefined features are considered as optional features regardless of their importance and urgency ranking.
- From the architectural point of view the IDE4OCL must provide interfaces to the tools in the OCL tools landscape (Fig. 1). Therefore the feature Interfaces to tools in the OCL landscape is also mandatory. An optional feature is Batch Mode.

All other predefined and proposed features we classify into features for *language and model support* and features for *user-friendly support*.

- Language and model support is declared as mandatory because one subfeature is OCL compliance that should be mandatory to achieve a high exchangeability with other tools. According to the OCL specification, at least one of the compliance points (syntax, XMI, evaluation) should be fulfilled. The predefined features (Association End Navigability, Name Resolution and Visibility and Lexical Scoping MOF) are aggregated into a compound feature for standardization support. Syntax and Semantics Extensions, Modularisation, OCL and Model Perspective, Variability as well as Syntax Highlighting are optional features for language and model support.
- The rest of the features we classify as user-friendly support that are all optional. To order them we distinguish feature for editing, reuse, maintenance and performance support. Editing support include typical editor features that are nice to have but not mandatory (Static Statement/Specification Analysis, Template Support, Symbol Database, Macro Mechanism, Code folding, Statement Element Browser, Collaborative Editing, Document Interface and Auto–indentation). Reuse Support was extended by two proposed features: Support of libraries and OCL testing. Statement Coverage, Version management, Error handling, and Documentation Specification and Generation are features that support maintenance of OCL packages. Besides a Profiler, OCL users wish Scalability of large models, large collections of objects and/or large OCL packages to improve the performance of OCL evaluation.

We want to explicitly note that the presented feature model is a first consolidated proposal that should be the subject of further discussions in the OCL community.

[20] The statistical evaluation of the appropriateness of the predefined features will be published in a paper on quantitative evaluation of the survey.

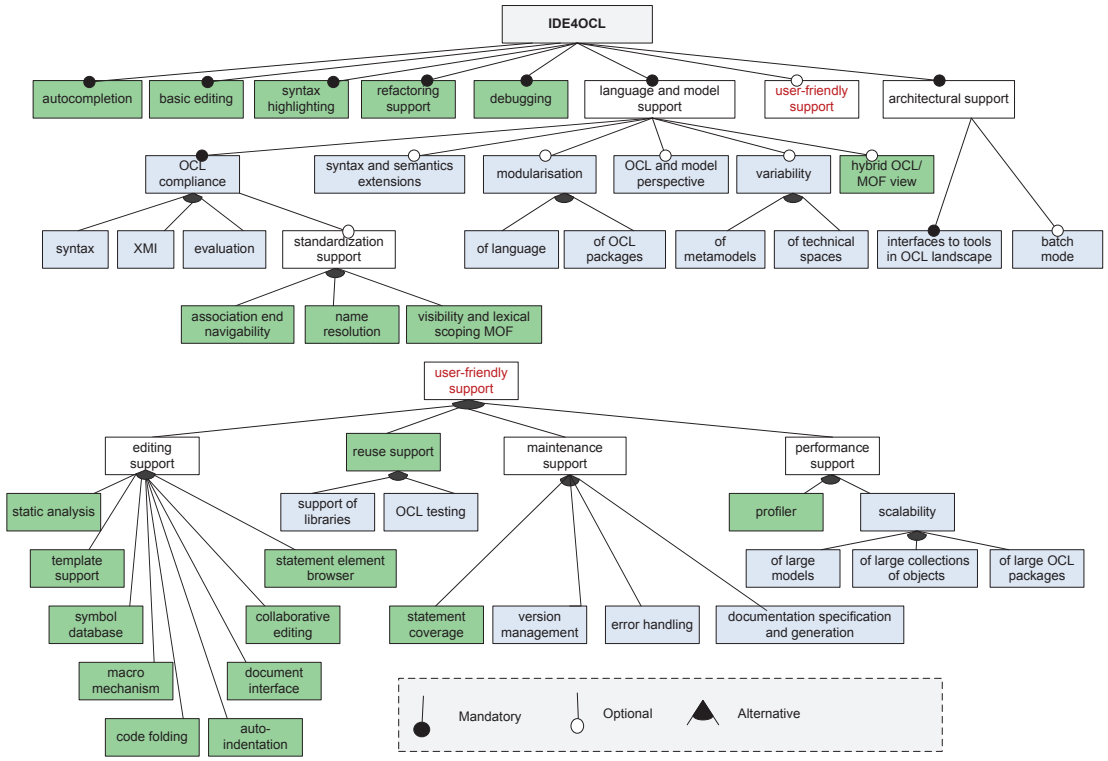


Figure 5: The IDE4OCL feature model with mandatory, alternative and optional features taken from predefined (in green) and proposed (in blue) features.

6 Conclusions and Future Work

Our long–term goal is the development of an OCL tool that can be part of a comprehensive model–driven software development environment. Additionally, we want to improve the usability of OCL usage for the software developer by efficient tool support. A requirements analysis is an important success factor in achieving this goal. Therefore we started with a solid requirements investigation for an IDE4OCL by questioning members of the OCL community both in academia and industry. We collected and analysed data, opinions and comments from around 100 respondents. Hence we may say that our results reflect the needs of a significant group of the OCL users.

In this paper we described qualitative feedback from the OCL community. We presented the most frequently mentioned features in the survey responses (Section 4.2). We created a proposal for a feature model of an IDE4OCL (Section 5) which includes predefined and proposed features. We recommend in a first version of an IDE4OCL the implementation of the features that are denoted as mandatory in the feature model. After a discussion in the OCL workshop we want to publish the updated feature model as a reference model and additionally as a comparison framework for OCL tools of IDE flavor.

In our future work we are looking for academic and industrial partners who are willing to collaborate in the further development, decomposition and implementation of our IDE4OCL vision within an open source project.

Acknowledgement The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsfoerderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT). Our gratitude goes to *all respondents of our survey*, especially to those who proposed additional features and gave us constructive feedback. Additionally, we want to thank *participants of the last OCL workshop*, *Manuel Clavel* and *Tricia Balfe* for interesting discussions. We would like to thank *Darius Silingas* and *Nicolas F. Rouquette* for co–operation in the initial phase of the project and *Kevin Church* and *Hannes Mösl* for their reviews during finalisation of the paper. Last but not least, our gratitude goes to *our student–developers* at universities of Innsbruck and Dresden for all discussions and their readiness to make our visions real.

Bibliography

[AZH08] D. Akehurst, S. Zschaler, G. Howells. OCL: Modularising the Language. In *Ocl4All: Workshop at MoDELS 2007, Electronic Communications of the EASST*. Volume 9. 2008.

[BW84] V. R. Basili, D. M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Trans. Software Eng.* 10(6):728–738, 1984.

[CCG+09] J. Cabot, J. Chimiak-Opoka, M. Gogolla, F. Jouault, A. Knapp. Ninth International Workshop on the Pragmatics of OCL and Other Textual Specification Languages. In Ghosh (ed.), *MoDELS Workshops*. Lecture Notes in Computer Science 6002, pp. 256–260. Springer, 2009.

[CDSR] J. Chimiak-Opoka, B. Demuth, D. Silingas, N. F. Rouquette. Requirements Analysis for an Integrated OCL Development Environment. *ECEASST* 24.

[CGG08] J. Cabot, M. Gogolla, P. V. Gorp. Eighth International Workshop on OCL Concepts and Tools. In Chaudron (ed.), *MoDELS Workshops*. Lecture Notes in Computer Science 5421, pp. 257–262. Springer, 2008.

[CO09] J. Chimiak-Opoka. OCLLib, OCLUnit, OCLDoc: Pragmatic Extensions for the Object Constraint Language. Pp. 665–669 in [SS09].

[Con09] L. L. Constantine. Interaction Design and Model–Driven Development. P. 377 in [SS09].

[CPP08] D. I. Chiorean, V. Petrascu, D. Petrascu. How My Favorite Tool Supporting OCL Must Look Like. *ECEASST* 15, 2008.

[CT09] J. Cabot, E. Teniente. Incremental integrity checking of UML/OCL conceptual schemas. *J. Syst. Softw.* 82(9):1459–1478, 2009. doi:http://dx.doi.org/10.1016/j.jss.2009.03.009

[DBR69] E. W. Dijkstra, J. N. Buxton, B. Randell (eds.). *Software Engineering Techniques*. 1969.

[DW09] B. Demuth, C. Wilke. Model and Object Verification by Using Dresden OCL. In *Proc. of the Russian-German Workshop Innovation Information Technologies: Theory and Practice*. 2009.

[HDF02] H. Hußmann, B. Demuth, F. Finger. Modular architecture for a toolset supporting OCL. *Sci. Comput. Program.* 44(1):51–69, 2002.

[HT99] A. Hunt, D. Thomas. *The pragmatic programmer: from journeyman to master*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Mel09] S. J. Mellor. Models. Models. Models. So What? P. 1 in [SS09].

[Rid84] W. E. Riddle. The magic number eighteen plus or minus three: a study of software technology maturation. *SIGSOFT Softw. Eng. Notes* 9(2):21–37, 1984. doi:http://doi.acm.org/10.1145/1010925.1010927

[SS09] A. Schürr, B. Selic (eds.). *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings*. Lecture Notes in Computer Science 5795. Springer, 2009.

[Sue06] J. G. Suess. Sugar for OCL. Proceedings of the Sixth OCL Workshop OCL for (Meta-)Models in Multiple Application Domains (OCLApps 2006), November 2006. In: Dan Chiorean and Birgit Demuth and Martin Gogolla and Jos Warmer (Eds.): Proceedings of the Sixth OCL Workshop OCL for (Meta-)Models in Multiple Application Domains (OCLApps 2006).

[WTW10] C. Wilke, M. Thiele, C. Wende. Extending Variability for OCL Interpretation. accepted for MoDELS 2010, 2010.

[WTZ10] C. Wende, N. Thieme, S. Zschaler. A Role–based Approach Towards Modular Language Engineering. In Brand et al. (eds.), *Software Language Engineering, 2nd Int'l Conf. (SLE 2009), Revised Selected Papers*. LNCS 5969, pp. 254–273. Springer, Mar. 2010.