Object Constraint Language

The Fallible Software Engineer.

Computers generally do not make mistakes. The CPU on the desks of millions the world over is perfectly capable of executing billions of instructions without making a single error. Why then do computers crash? We already know the short answer - software bugs, which are introduced by the programmer into the software unwittingly and/or accidentally during its construction. Bugs are what result when the programmers intent differs from the actual implementation as defined by the code; the code always has the final say on the existence of bugs.

A programmers job, in the loosest sense, has always been to translate an "idea" into executable code. The idea can be described by the programmer (or software architect, or software engineer) in many ways: by using diagrams, by using the written word, even by using the spoken word, along with para-linguistic features such as hand gestures. However, none of these forms is of any use to the computer, and so the idea must be eventually be translated into a language understood by both programmer and computer; a programming language.

It is common knowledge that the adoption of progressively higher level programming languages over the last few decades has reaped huge rewards in terms of the quality of software and the productivity of programmers. Until recently however, the translation of the "idea" to the programming language of choice has remained a largely manual process, undertaken by fallible software engineers.

The solution to this problem which is being championed by the Object Management Group (OMG) is the Model Driven Architecture(MDA). If the software "idea" can be expressed with enough precision then the process of translation from idea to programming language can be automated. This automation will eliminate errors which would have previously been introduced at this stage by the programmer. The difficulty lies in expressing the "idea" with enough precision for this automation to become feasible.

Software Modelling and the UML

The process of specifying a software "idea" is more commonly known as modelling. Software modelling has been a research area in computer science for many years but has only recently begun to enter into common usage with the standardisation of the Unified Modelling Language (UML1.1) in 1997. The OMG defines UML as the standard modelling language for object oriented modelling, including non-software (i.e. business process) modelling. The core of UML1.1 focuses on "diagrammatic elements and giving meaning to those elements through English text" [Warmer 2003, pg. xx] . Diagrams and the written word are favoured by humans for expressing ideas. However the OMG recognizes the limitations of UML diagrams in constructing a model with sufficient

precision for the automatic translation goal of the MDA. The precision of UML is provided by the use of a formal language, the Object Constraint Language.

Formal Languages

Programming languages used in computer science are always formal languages. Mathematicians have constructed and used formal languages for many years to talk precisely about a given area of mathematics an example being the lambda calculus. A formal language is by definition "a mode of expression more careful and accurate, or more mannered than everyday speech." [Wikipedia, Formal Languages]

In addition to programming languages, formal languages have been used in the field of computer science to describe programs in a more abstract sense than the nitty-gritty of an ordered list of instructions. For example, the language Z was developed at Oxford University in the 1970s to specify and formulate proofs about computer programs. Later after the advent of object oriented techniques and programming languages the language Syntropy was created which was partly built on a subset of Z. [Wikipedia, Z notation]

Formal languages provide something that our natural language doesn't, a single interpretation of a given statement. The English language for example, though enormously expressive, is inherently ambiguous. For example, the phrase "I saw the flowers walking down the garden path." either describes a pleasant stroll, or a horror B-movie. Formal languages exist for their lack of ambiguity.

The Object Constraint Language is the formal language for modelling with the UML. It provides the unambiguous expression that UML diagrams alone cannot provide.

A formal language such as Z or Syntropy can be used for precise modelling of software. Traditionally formal languages have used a mathematical notation which often leads to the language being inaccessible to non-mathematicians "Experience with formal or mathematical notations have led to the following conclusion: The people who can use the notation can express things precisely and unambiguously, but very few people can really understand such a notation" [Warmer 2003, pg. 17]

OCL was designed from the beginning to be accessible to a wider audience than previous modelling notations. It contains no unfamiliar mathematical symbols. It is also easy to learn; "OCL was very carefully designed to be both formal and simple: its syntax is very straightforward and can be picked up in a few minutes by anybody reasonably familiar with modelling or programming concepts” [Warmer 2003. pg. Xix]. These facets have helped OCL already gain a much wider acceptance in industry than previous formal modelling languages.

Constraints

OCL is used to augment the power of the UML. In the initial UML1.1 specification, the OCL was defined as an extension to the language. Additionally, this original OCL

specification focused on one particular usage of the OCL, that of constraints. Briefly "a constraint is defined as a restriction on one or more values of (part of) an object-oriented model or system" [Warmer 2003, pg.xxii].
The OCL2.0 specification extends the use of OCL to more generalized expressions; "In UML 2, the understanding is that far more additional information should be included in a model than constraints alone. Defining queries, referencing values, or stating conditions and business rules in a model are all accomplished by writing expressions." [Warmer 2003, pg.3].

Alternative syntaxes

As we have mentioned earlier, the syntax of OCL contains no unfamiliar mathematical symbols, making the language accessible to a wide audience. OCL goes even further than this to accommodate different usage patterns; the OCL2.0 specification allows users to define their own syntax. For instance, a particular usage domain may lend itself to an alternative syntax, one example of this is the Business Modeling syntax [Warmer 2003, Appendix C]. The new syntax must be able to be mapped onto the standard syntax. For example Klasse Objecten provide a tool to transform standard OCL syntax to BM syntax, known as Octupus [Klasse Objecten].

Models Only

"Precise specifications are necessary when those specifications are meant to be realized by a computer, since most computers do not tolerate ambiguity" [Clark 1998, pg.2]. The ability to create models that communicate their intent unambiguously is key problem on the path to the realization of the MDA. The OCL is the OMGs solution to this problem. With the OCL UML models can be defined to a level of precision necessary for automatic translation Such tools have already emerged, the Klasse Objecten website lists the current offerings [Klasse Objecten].

The OMG defines five maturity levels related to the use of models within software development. Level five, the uppermost level, is described as 'models only'; the goal of software development from a model, with no user intervention in actual the construction of executable code. "This level has not been realized yet anywhere in the world. This is future technology, unfortunately." [Warmer 2003, pg.12]. The OCL2.0 is another step on the path towards this goal.

References

Clark, Tony and Warmer, Jos, 1998, Object Modeling with the OCL – The Rationale behind the Object Constraint Language, Springer, Berlin.

Warmer, Jos and Kleppe, Anneke, 2003, The Object Constraint Language Second Edition – Getting Your Models Ready For MDA, Addison-Wesley, Boston.

Wikipedia, Formal Language, http://en.wikipedia.org/wiki/Formal_language
Wikipedia, Z notation, http://en.wikipedia.org/wiki/Z_notation
Klasse Objecten, www.klasse.nl/ocl