International Journal of Innovation Scientific Research and Review
Vol. 03, Issue, 02, pp.844-846, February, 2021
Available online at http://www.journalijisr.com

Research Article **ISSN: 2582-6131**

LANGUAGE INDEPENDENT METAMODELS FOR MODELLING OF OBJECT ORIENTED SOFTWARE

[1], *Ajit kumar and [2]Dr. Navin Kumar.

[1]Research Scholar, Dept. of BCA, S.R.K.G. College Sitamarhi, B.R. Ambedkar Bihar University, Muzaffarpur, India.
[2]Faculty Member, Dept. of BCA, M.D.D.M College, B.R. Ambedkar Bihar University, Muzaffarpur, India.

Received 28th December 2020; Accepted 22th January 2021; Published online 15th February 2021

ABSTRACT

A language independent Meta Model supports the object oriented model on the basis of requirement. An environment for reengineering and reverse engineering should be extensible in three aspects. First the repository should be able to represent and manipulate entities other than the one directly extracted from source code. Second the reengineering environment should be able to operate with external tools like diagrammers, parser, and finally graphics drawing tools and to support reengineering in the context of software evolution. The environment should be able to handle several source code models simultaneously. The old system contains a large number of entities, so it should be scalable in terms of new system. The tools specific to the reengineering environment should support the programming languages like C++, Ada, Java and Smalltalk. A re-engineering effort is typically a cooperation of a group of specialized tools. Therefore, a re-engineering environment needs to be able to integrate with external tools, either by exchanging information or ideally by supporting runtime integration. An additional requirement in this context is the actual performance of such an environment. It should be possible to handle a legacy system of any size without long latency times.

Keywords: Language independent Meta Model, Reengineering, Repository, Integration.

INTRODUCTION

A language independent Meta Model supports the object oriented model on the basis of requirement. The requirement of language independent Meta-Meta Model is as follows
Extensibility: An environment for reengineering and reverse engineering should be extensible in three aspects. First the repository should be able to represent and manipulate entities other than the one directly extracted from source code. Second the reengineering environment should be able to operate with external tools like diagrammers, parser, and finally graphics drawing tools and to support reengineering in the context of software evolution. The environment should be able to handle several source code models simultaneously. **Scalability:** The old system contains a large number of entities, so it should be scalable in terms of new system.
Support for multiple languages: The tools specific to the reengineering environment should support the programming languages like C++, Ada, Java and Smalltalk. **Information exchange:** A reengineering effort is typically a cooperation of a group of specialized tools [*Serge Demeyer et al., 1999*]. Therefore, a reengineering environment needs to be able to integrate with external tools, either by exchanging information or ideally by supporting runtime integration. In addition to these general requirements, the context of the FAMOOS project [*Stéphane Ducasse and Serge Demeyer, 1999*], in which Moose was originally developed, imposed the following requirement: **Scalable:** As legacy systems tend to be huge, an environment should be scalable in terms of the number of entities being represented. Furthermore, it should provide meaningful information at any level of granularity. An additional requirement in this context is the actual performance of such an environment. It should be possible to handle a legacy system of any size without long latency times. **Support for multiple object-oriented languages:** This specific tool environment must support the reengineering of software systems written in C++, Java, Ada and Smalltalk.

***Corresponding Author: Ajit kumar,**
[1]Research Scholar, Dept. of BCA, S.R.K.G. College Sitamarhi, B.R. Ambedkar Bihar University, Muzaffarpur, India

EXPERIMENTAL DETAILS

The FAMIX Meta-Meta Model models multiple object-oriented languages. The design space as. FAMIX supports multiple languages within one paradigm. It defines a language - independent core, which allows tools to be reusable without adaptation over the supported languages. How languages are mapped to the core and which language specifics can be stored, is specified in language extensions. Here i present the core part of FAMIX. The Meta-Meta Model represents source code at the program entity level. First of all, this level of information is sufficient for the analysis tasks I want to support. The information allows one to perform structural analysis and dependency analysis. It supports metrics computation and heuristics. It does not support control flow analysis and the regeneration of source code from the model. I store, however, the location of the source code, allowing one to obtain additional information from the source code itself. The second reason to choose the program entity level is that more detailed information increases the size of models considerably which hampers scalability. Thirdly, the program entity level enables to abstract from language- specific details and as such allows for a clean language-independent Meta-Meta Model. Figure1 shows the core entities and relations. All basic elements of object-oriented languages are present (Class, Method, and Attribute). Furthermore, FAMIX models support dependency information, such as method invocations and access of the attribute and method accesses. This information is important for dependency, impact analysis and for instance.

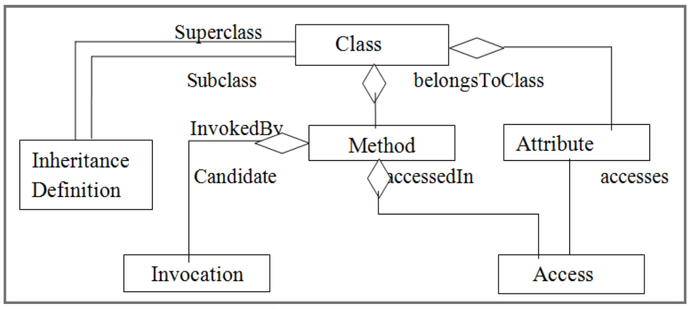

Figure1. Core entities and Relations

The complete Meta Model is not restricted to the above elements. Additionally it also models different kinds of variables, functions and arguments. I have given a short description of these elements. They are modeled in an object-oriented hierarchy, which is shown in Figure 2 and figure 3.

- Function: a definition of behavior with global scope.
- Local Variable: it is a variable that is local to a method or function.
- Global Variable: it is a variable with global scope.
- Implicit Variable: variables that are not explicitly defined such as self, this and super.
- Formal Parameter: a parameter of a method or function.
- Access Argument: an argument of an invocation that constitutes a simple variable access.
- Expression Argument: an argument of an invocation which is an expression.
- Package: a scoping mechanism.
- Model: a Meta entity containing information about a model such as creation time.

XMI Meta-Meta Model, Functions and global variable are modeled because they exist in several object-oriented languages I want to cover such as C++ and Smalltalk. This effectively makes FAMIX support hybrid object-oriented and procedural languages.

Object-Oriented Hierarchy	• Function: • Local Variable • Global Variable. • Implicit Variable • Formal Parameter • Access Argument • Expression Argument • Package. • Model

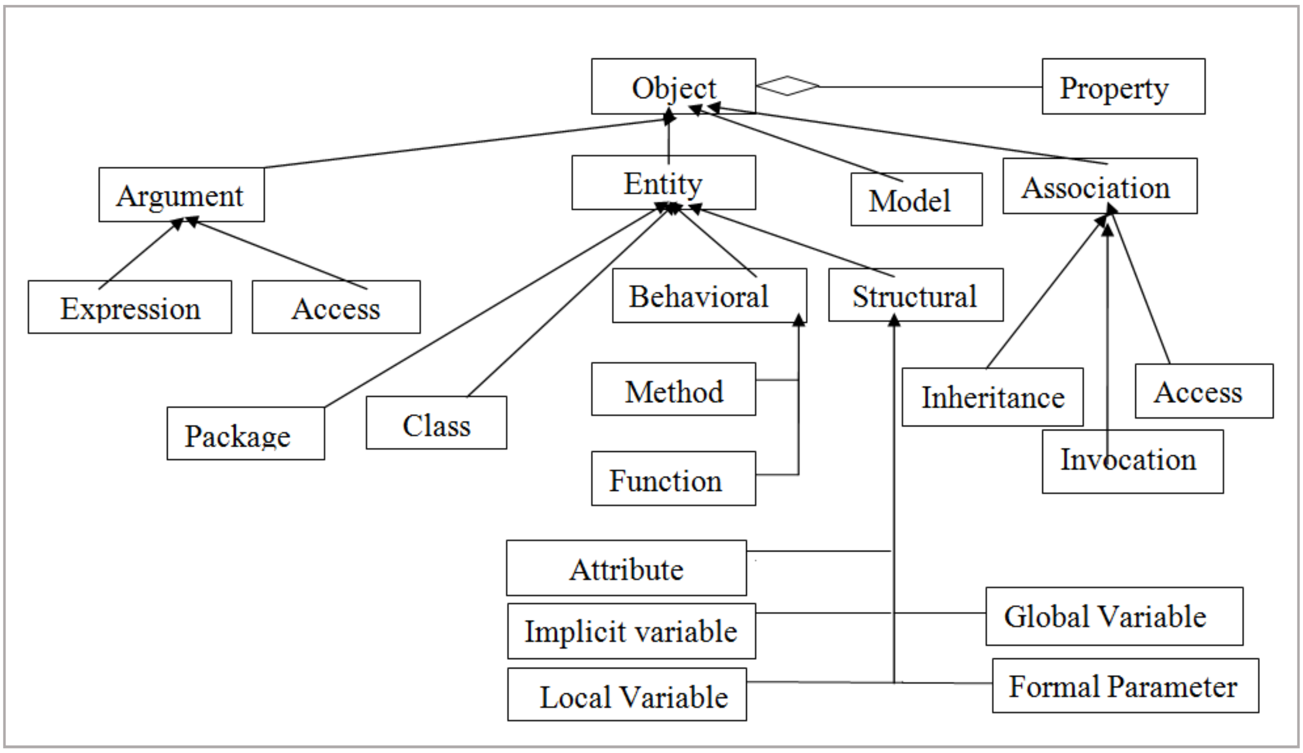

Figure 3. Hierarchy of Famix model

New model elements

An extension can define new model elements. Examples are the Include relationship for the C++ extension [Stéphane Ducasse and Serge Demeyer, 1999] and the Measurement element for the metrics extension.

New attributes to existing model elements

Existing elements can be extending to allow one to store additional information. An example is the is Final attribute that the Java extension adds to the definition of the Method element. One of the problems I have encountered is that not all standard Meta-Meta Model support class extension. In the context of textual information exchange I have worked with CDIF [*CDIF, 1999*] and XMI [OMG, 1997]. The CDIF Meta-Meta Model supports class extensions, the

Annotations

Any model element can be annotated by attaching a Property to it. This is shown in figure 4 by the Object class, which can have zero or more Properties attached to it.

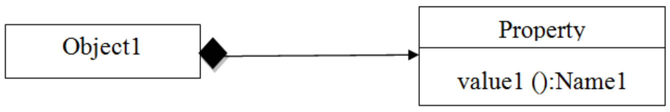

Figure 4. Famix model elements and their Annotation

Multiple Language Support Provided by FAMIX

The FAMIX Meta Model supports multiple object-oriented languages.

Here I have described the design decision that makes it easier to support more than one specific language.

Concept of General Multiple Language Design

In FAMIX Meta Model we find relevant information for the support of multiple languages.
Multiple Inheritances
FAMIX supports multiple inheritances. This allows us to deal with single inheritance languages such as Smalltalk, but also with multiple inheritance languages such as C++. Java also fits this scheme by interpreting Java interfaces as abstract classes and interface implementation as common inheritance.

Statically typed and dynamically typed languages

Static type information is important to store, because it reveals important dependencies. If the information is not known, which is normally the case with dynamically typed languages such as Smalltalk, the information is left empty. For instance, Figure4 shows that Method inherits the declared Type attribute. It is used to store the statically declared return type for methods, like Point for the method declaration Point get Point () {...} in Java. It is left empty for Smalltalk methods. Another example of supporting both dynamic and static typing is the candidate methods of an invocation.

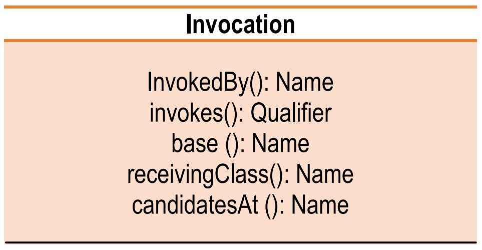

Figure 5 shows the Invocation entity. The candidates attribute stores the methods possibly invoked by this invocation. In Smalltalk, without static type information, the candidates are all methods in a system that have the signature as stored in the invokes attribute1. In Java the static type information reduces the possibly invoked methods to a single inheritance hierarchy or interface implementation hierarchy. By storing the candidates independent of the way the information is collected, tools can use the information independent if it concerns a dynamically or statically typed language. Application: language independent Meta Model supports various object oriented programming like C ++, Java, Ada and Smalltalk. Static type information is implemented in java programming language in case of single inheritance. In Smalltalk, without static type information, the candidates are all methods in a system that have the signature as stored in the invokes attribute.

Conclusion

FAMIX supports multiple inheritances. This allows us to deal with single inheritance languages such as Smalltalk, but also with multiple inheritance languages such as C++. Java also fits this scheme by interpreting Java interfaces as abstract classes and interface implementation as common inheritance.

REFERENCES

1. [DD99] Stéphane Ducasse and Serge Demeyer, editors. The FAMOOS Object-Oriented Reengineering Handbook. University of Berne, October 1999.
2. [DDL99] Serge Demeyer, Stéphane Ducasse, and Michele Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, Proceedings WCRE'99 (6th Working Conference on Reverse Engineering). IEEE, October 1999.
3. [Neb99] Robb Nebbe. FAMIX Ada language plug-in 2.2. Technical report, University of Berne, August 1999.
4. [Fre00] Michael Freidig. XMI for FAMIX. Informatikprojekt, University of Berne, June 2000.
5. [TD99] Sander Tichelaar and Serge Demeyer. SNiFF+ talks to Rational Rose– interoperability using a common exchange model. In SNiFF+ User's Conference, January 1999.
6. [Com94] CDIF Technical Committee. CDIF framework for modeling and extensibility. Technical Report EIA/IS-107, Electronic Industries Association, January 1994
7. [DDT99] Serge Demeyer, Stéphane Ducasse, and Sander Tichelaar. Why unified is not universal. UML shortcomings for coping with round-trip engineering. In Bernhard Rumpe, editor, Proceedings UML'99 (The Second International Conference on The Unified Modelling Language), LNCS 1723, Kaiserslautern, Germany, October 1999. Springer-Verlag.
8. [Bar99] Holger Bar. FAMIXC++ language plug-in1.0. Technical report, University of Berne, September 1999.
9. [OMG97] Object Management Group. Meta object facility (MOF) specification. Technical Report ad/97-08-14, Object Management Group, September 1997.

.

****