Translating semantic networks to UML class diagrams

Ivan Barashev*

Higher School of Economics, 20 Myasnitskaya Ulitsa, Moscow 101000, Russia

Abstract

The high importance of a proper definition of software architecture is obvious, but this task is not easy, especially in large projects. The project scheme should be legibly structured and clearly visualized. To that aim, usually widespread UML class diagrams are used. However, this way is not flexible and clear enough at the initial state of the development, so there is a need of a more appropriate method. Semantic networks are not so commonly used to meet these demands. Nevertheless, this notation is a fairly suitable approach to building evolutionary computing system architectures. Also, this can be a good mediator for other formats. Moreover, the tools for that may become quite powerful and relevant.

Keywords: UML class diagrams; semantic networks; software architecture

* Corresponding author. E-mail address: aivanf@mail.ru

1. Introduction

The more advanced computer systems become, the more difficult it is to work with them. However, a programmer cannot keep in mind all the picture of the application he is developing because human memory does not increase and stays the same. Hence, the elaborated systems must be well organized to avoid extra thinking about different details. Also, eventually this will allow to avoid plenty of problems. That is why software architecture is so important. Therefore, the tools for building software architecture are so important. As Revees wrote: “I would say we need good architectures — top level design, good abstractions — class design, and good implementations — low level design.” [5]

A number of studies and guides of software architecture definition find UML diagrams an imperfect tool and recommend do not to use them at the beginning of the work [1-3]. Unnecessary formalism makes development more complicated rather than clear. The majority of other object-oriented tools have the same difficulty: they provide their
own strict rules and rigid frameworks. In addition, there is almost no tool to connect different formats with others. Hereby, there is a need of more flexible method to build application architecture. In addition, the notation of semantic networks can be used for that: it is quite flexible and any other notation can be described using it.

2. Background and related works

There are several well-known kinds of diagrams which are used to represent and describe object-oriented systems. Here they are:

- Unified Modeling Language (UML) diagrams and, specifically in this paper, class diagrams. A number of popular IDEs can build UML class diagrams from an existing source code to improve visualization and documentation. Some special tools allow to make class diagrams using drag and drop interface, and others can create templates of code using the given class diagrams. UML specification also provides various types of diagrams with other rules which help to describe systems from different object-oriented views. Furthermore, it is widely used in industry for a long time [7].
- Entity-relationship (ER) model. It is also used to describe objects of some systems, mostly relational databases. It is used in some popular applications, in instance, “Microsoft Visio”.
- Ontologies. Applications such as famous “Protégé” and “OWLGrEd” allow to develop systems of ontologies which are generally based on frames. Frames are used in the artificial intelligence field to describe relationships between objects.

These notations are widely used in various situations by small and large IT teams. And they have been in demand for a several decades. However, they have a few disadvantages which disturb software development process:

- These formal notations are too strict, focused on low-level aspects of concepts and associations. Thus, they are not yielding enough to high-level influence. However, such formats either cannot describe every facet of application, only the code is capable of it. As Reeves said, “the code is the design” [5]. For the same reason, such methods of representation are not scalable. Moreover, large systems contain dozens of objects which can and should be merged together in higher scale of view.
- UML specification is difficult to study and use. It contains a great deal of diagram kinds with their own rules. On the one hand, UML class diagrams describe a lot of details of systems which are not so necessary at the beginning of architecture definition. Hereby, it is also time-consuming. On the other hand, it lacks a formal definition, which is necessary for large projects [8]. Some of these and other difficulties of UML are discussed by Bachmann et al [1] and Avgeriou et al [3].
- There is no converter between described formats. Although it is used quite rarely, there are some situations where it is necessary. Nevertheless, some attempts were made to connect UML diagrams with ontologies [7,8].

For these reasons, some developers try to create and use new methods for architecture building and representation which are more flexible, understandable and suitable.

The project called “Structurizr” was developed as an alternative to UML diagrams and a few applications for software definition. This provides its own simple rules and application with drag and drop interface for easy building software architecture. The necessity this tool is explained by Brown [2]. However, the server part of the application is chargeable, and it does not interact with different notations.

The developers of “Semantic Research Inc.” created “Semantic Object Language” (SOL) which is based on a semantic network. The key advantage of this graphical format is the representation of the system in relation to one selected object. While it represents object systems in a very dynamic and readable way, this format does not provide an exact way of architecture construal. Thus, SOL has not become popular.
3. Problem formulation

Given that development of large projects is a complex process, software engineers use some notations to describe developing systems in order to represent them visually. Often rigid notations are used for that, however, they are not flexible enough. In addition, most of formats are not compatible. Meanwhile, semantic networks are quite simple and yielding to influence. According to Sowa [4] and Osipov [6], UML and frames are examples of hybrid of multiple semantic network notations. Thus, semantic networks can easily become a mediator for other notations.

The problem of this work is to link class formalism with a free conceptual notation. In other words, the goal is to create a tool for translating strict UML-diagrams of classes to semantic networks and backwards in order to use the flexible network notation at first stages of projects development. Hence, it will allow to elaborate evolutionary software architecture. For example, it is possible to begin the project with a mind map, then to develop it into a complete semantic network and eventually to design a class diagram and a code template. The key attribute of this tool is transparency, which means that the tool must have visualization.

There is no common textual notation for UML class diagrams, therefore a good way to represent them textually is to use a programming language. Furthermore, the most widely used textual notation for semantic networks is a functional one.

4. Practical results

The above mentioned program has been developed for two operating systems: Windows and Mac OS X. This program can convert a code of functional notation of semantic networks to code in the C# programming language and backwards. The application interface contains two text fields for input and output, combo boxes for formats choice and a button for text processing. In addition, UML class diagram and semantic network graph can be drawn for visual perception by pressing the relevant buttons below each text field. The figure below shows the main window of the program.

![Application screenshot.](image)

At the beginning, the parser of functional notation of semantic networks gets a list of functions with two arguments, which describe directed links between concepts. The application provides a few of standard relationships
types (such as: “is a”, “has variable”, “of type”), which are available for converting to a programming language, but it can also process any other names of links. The parser successfully detects if parentheses are parts of a method name and distinguish different nodes with the same name. In addition, detected mistakes of the input text are printed with their line numbers and a comments.

Furthermore, the application was written in C# using Visual Studio for Windows and Xamarin for Mac OS. Its main part is a library for text process in different formats. This library is the same for both operating systems. Interaction with it is based on Model-View-Controller software design patterns. The controller is core controller class, the view is window controller interface, and the model is core class with nodes and links. In addition, internal organization of the library is based on Factory pattern. In this way, the core controller uses different parsers, code generators and visualizers, which are chosen considering selected by user types.

To draw graphs, the library visualizers create scripts of DOT language. Then, the application sends it to the libraries of “GraphViz” program. “GraphViz” is a powerful, popular, cross-platform and freeware tool that can draw any kind of graph or diagram and is able to handle hundreds of nodes and links. It also provides a great deal of shapes and attributes for nodes and arrows.

5. Results of experimental verification

The developed instrument was tested within a few university small and middle-scale projects. It was used to define and concretize significant aspects of applications architecture and behavior. Also, it has proven a useful analytical tool when some major changes of architecture were required. However, it has one remarkable drawback: the developed tool does not provide the drag and drop interface, that is not convenient.

The second figure illustrates a semantic network visualized using the instrument. As well as the previous figure, it shows a part of architecture of program developed using our tool. In addition, shapes of arrows can be easily changed using special integrated functions.
6. Conclusion

The definition of application architecture is one of the major challenges related to large projects development, since instruments for building systems are an indispensable part of software engineering. There is a number of variety methods for architecture building with surplus formalism. One of the possible solutions to this problem is to connect and convert different formats and to use a flexible notation with formals. The connection of class diagrams with semantic networks has revealed that semantic networks provide the necessary plasticity for the development and this notation is a proper mediator for other formats.

In the future, the developed program can be improved by including other languages and notations (for example, ontologies, C++ programming language or conceptual semantic networks), making it more interactive and introducing the drag and drop interface. Perhaps, this application will not become very popular, but it is a step in such important field, and this experience can be used in future projects.

References